

**ENGINEERING @ SP**



**ET0529 / ET0530 / EM0413**

**MOBILE APPLICATIONS /  
MOBILE APPS /  
MOBILE APP /  
DEVELOPMENT**

**(Version 2.5)**

**School of Electrical & Electronic Engineering**

# ENGINEERING @ SP

## The Singapore Polytechnic's Mission

As a polytechnic for all ages  
we prepare our learners to be  
life ready, work ready, world ready  
for the transformation of Singapore

## The Singapore Polytechnic's Vision

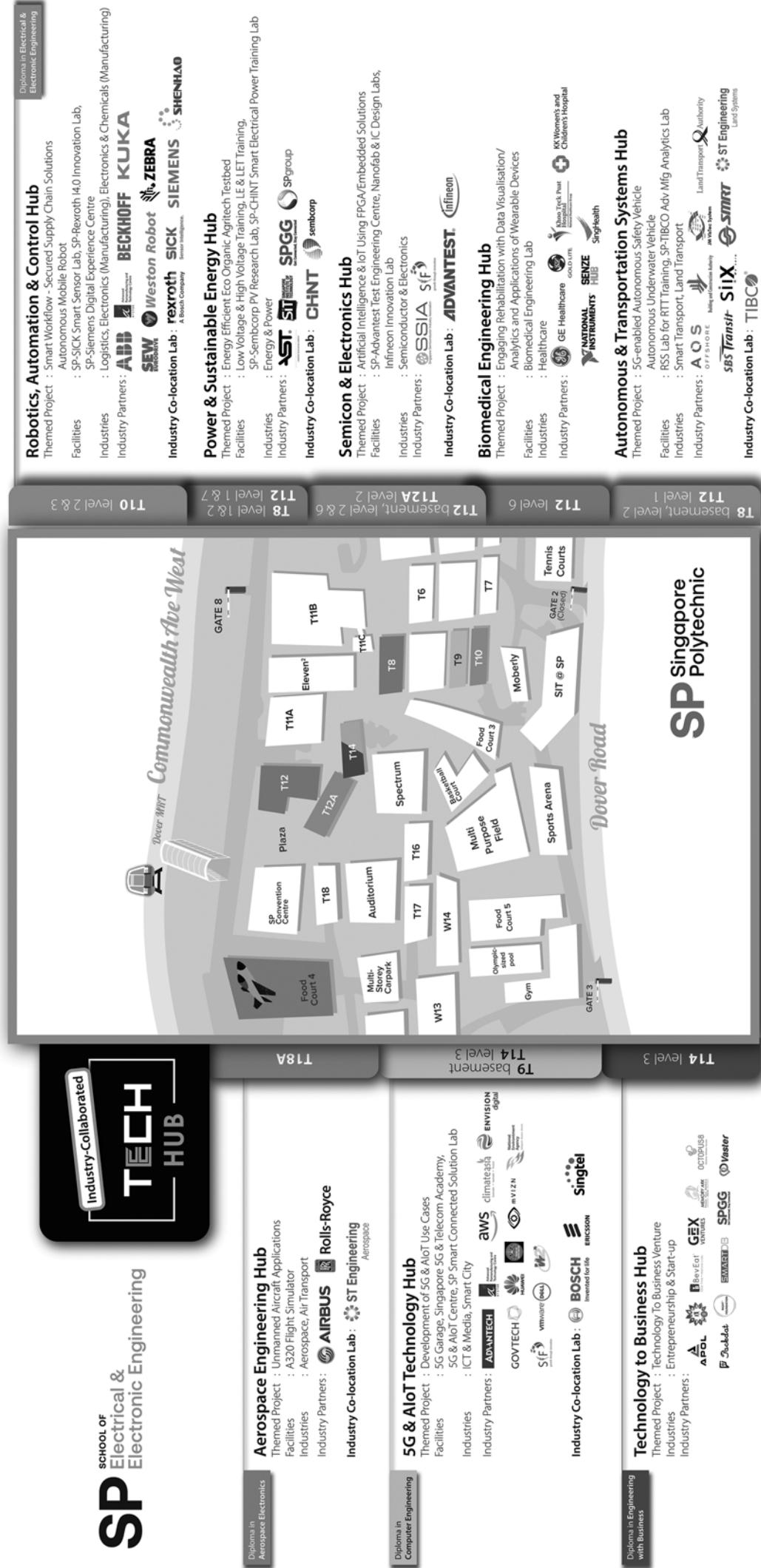
Inspired Learner. Serve with Mastery. Caring Community.  
A caring community of inspired learners committed to serve with mastery.

## The SP CORE Values

- Self-Discipline
- Personal Integrity
- Care & Concern
- Openness
- Responsibility
- Excellence

For any queries on the notes, please contact:

Name: K Y Wong  
Room: T947  
Email: WONG\_Kwee\_Yin@sp.edu.sg  
Tel: 68790682



| <b>CONTENTS</b> |   |               |
|-----------------|---|---------------|
|                 |   | <b>Page</b>   |
|                 | <b>Module Overview</b>  | 1             |
| <b>Chapter</b>  | <b>Topic</b>  |               |
| 1               | Introduction –Session 1 Overview of Android                           | Ch1S1. 1 – 36 |
|                 | Introduction –Session 2 Overview of Object Oriented Programming (OOP) | Ch1S2. 1 - 22 |
| 2               | Getting Started   | Ch2. 1 – 28   |
| 3               | Creating User Interfaces  | Ch3. 1 – 35   |
| 4               | List & Tab View   | Ch4. 1 - 24   |
| 5               | Menu & Database   | Ch5. 1 - 19   |
| 6               | Activities  | Ch6. 1 - 19   |
| 7               | Location & Map  | Ch7. 1 – 30   |
| <b>Lab</b>      |   |               |
| Getting Started | Install IDE and Create Your First App                                 | L.1-13        |
| Lab 1           | Form  | L1. 1 – 25    |
| Lab 2           | List View & Tab View  | L2. 1 – 18    |
| Lab 3           | Menu & Database   | L3. 1-- 18    |
| Lab 4           | Activities  | L4. 1 – 18    |
| Lab 5           | Location & Map  | L5. 1 – 22    |

\*Please refer to Brightspace | Additional Resources for further reading

## **Module Overview**

### **1 Introduction**

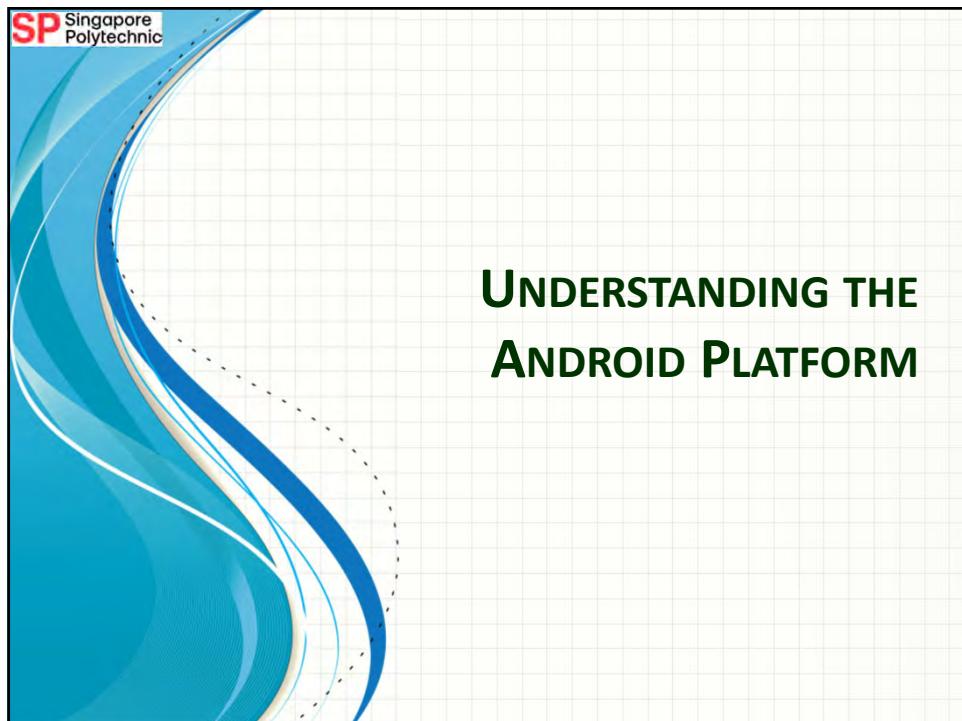
Have you ever wondered how your favourite mobile applications are developed? Join us on a journey through the mobile application development landscape, using Android as the platform. This module is intended for students who have some prior programming experience. It will introduce you to the basics of the Android platform, Android application components, Activities and their lifecycle and UI design in Android.

### **2 Module Aims**

This module provides students with the skills and knowledge to develop and implement applications that can run on mobile devices. Students will be introduced to open-source software tools available for program development, key concepts in mobile application programming, user-interface design, network connection and persistent storage. By the end of the module, students should be able to apply and develop a mobile application.



1



2

## Today's Overview

1

- What is Android

2

- Android System Architecture Overview

3

- Android Application Architecture Overview

4

- Function of Intent & IntentFilter

## What is Android?

- Android is a **software environment for mobile devices (not a hardware platform)** that includes an operating system, middleware and key applications
- Android is **not a particular device**, or even class of devices. It is a **platform that can be used and adapted to power different hardware configurations**

## Android Devices

- **Mobile phones** are the main class of Android powered devices. Other android powered device examples include **electronic book readers, netbooks, tablets, and set-top boxes, etc.**



5

# System Architecture Overview

6

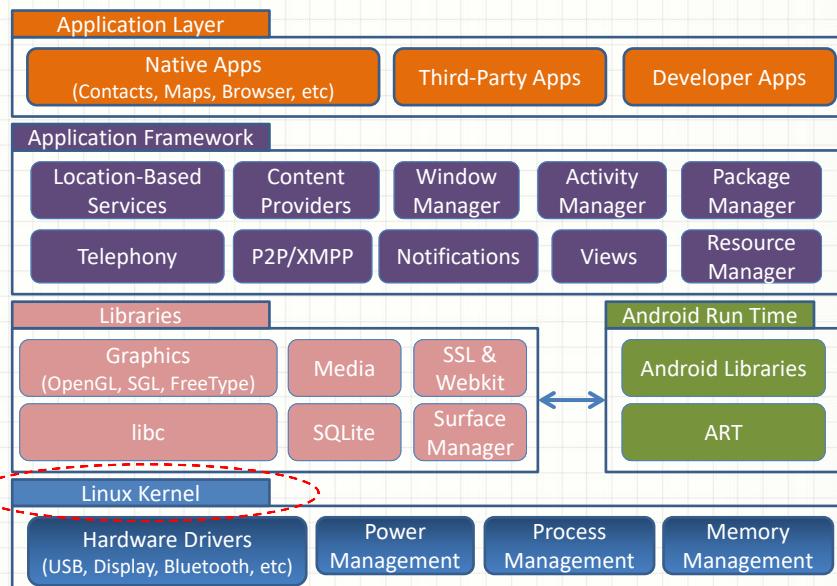
# Android System Architecture

- an Android system comprises of **4 basic layers**
  - Linux Kernel
  - Libraries & Android Runtime
  - Application Framework
  - Applications



7

# Android System Architecture



8

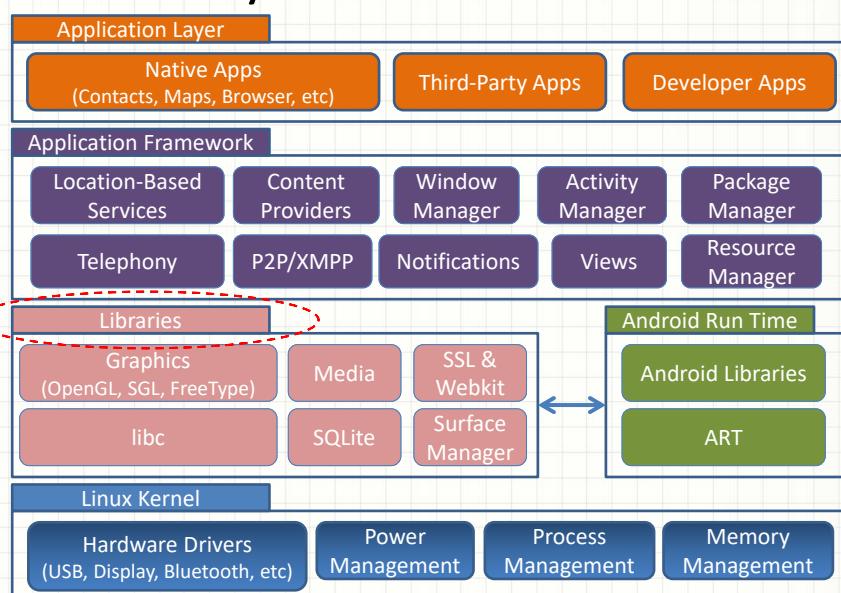
# Android System Architecture

## Linux Kernel

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

9

# Android System Architecture



10

# Android System Architecture

## Libraries

- Running on top of the kernel, Android includes
  - C/C++ core libraries
    - such as libs and SSL
  - Media library
    - for playback of audio and video media
  - Surface manager
    - to provide display management

11

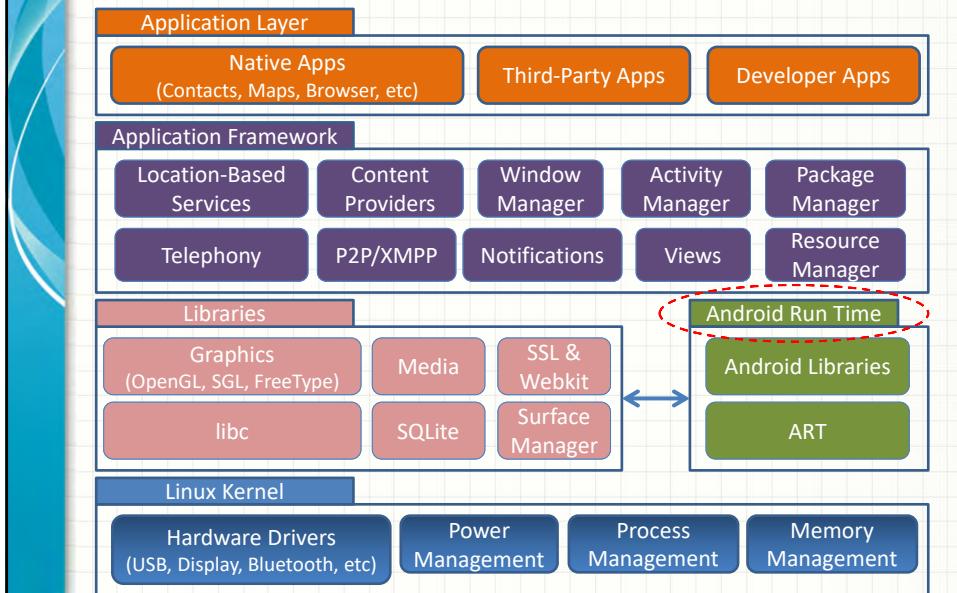
# Android System Architecture

## Libraries

- Running on top of the kernel, Android includes
  - Graphics libraries
    - include SGL and OpenGL for 2D and 3D graphics
  - SQLite
    - for native database support
  - SSL and WebKit
    - for integrated browser and Internet security

12

# Android System Architecture



13

# Android System Architecture

## Android Run Time

- The Android run time is the engine that executes your applications and, along with the libraries, forms the basis for the application framework

14

# Android System Architecture

## Android Run Time

Including

- **Core libraries**

The core Android libraries provide most of the functionality available in core Java libraries as well as the Android-specific libraries

- **ART (Android Runtime)**

Android Runtime (ART) is an application **runtime environment**.

It performs the **translation** of the application's **bytecode** into **native instructions** that are later executed by the device's runtime environment.

15

# Android System Architecture

## Android Run Time

- **Android Runtime (ART)**

Android app is developed using the **Java** or **Kotlin** programming language. It is compiled to the Java **bytecode** format.

ART replaces the original **Dalvik** virtual machine for devices running Android version 5.0 (API level 21) or higher.

Dalvik is based on **JIT** (just in time) compilation. It means that each time you run an app, the part of the code required for its execution is going to be translated (compiled) to machine code at that moment. As you progress through the app, additional code is going to be compiled and cached, so that the system can reuse the code while the app is running. Since JIT compiles only a part of the code, it has a smaller memory footprint and uses less physical space on the device.

16

# Android System Architecture

## Android Run Time

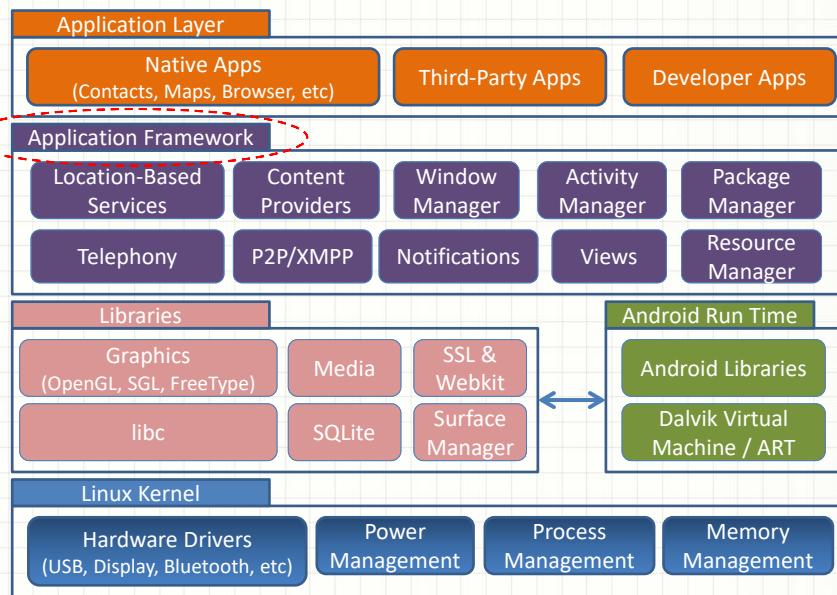
- **Android Runtime (ART)**

ART introduces the use of **ahead-of-time (AOT)** compilation by compiling entire applications into native machine code upon their installation. By eliminating Dalvik's interpretation and JIT (Just-In-Time) compilation, ART improves the overall execution efficiency and reduces power consumption. ART brings faster execution of applications.

As a downside, ART requires additional time for the compilation when an application is installed, and applications take up slightly larger amounts of the storage space (flash memory) to store the compiled code.

17

# Android System Architecture



18

# Android System Architecture

## Application Framework

- A set of high-level building blocks for creating extensive, interactive, rich graphical apps, and is targeted to deploy these apps to the Google Play Store.
- It is available through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services.

19

# Android System Architecture

## Application Framework

The **preinstalled framework on Android devices** consists of the following components:

- **Activity Manager**  
manages the lifecycle of apps and maintains a shared activity stack for navigating within and among apps
- **Views**  
A rich and extensible **View System** you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser

20

# Android System Architecture

## Application Framework

- **Notification Manager**  
enables all applications to display custom alerts in the status bar
- **Content Providers**  
enable apps to access data from other apps (such as Contacts), or to share their own data
- **Resource Manager**  
providing access to non-code resources such as localized strings, graphics, and layout files

21

# Android System Architecture

## Application Framework

- **Location Manager**  
enables an Android device to be aware of its physical location
- **Package Manager**  
lets an app learn about other app packages that are currently installed on the device
- **Telephony Manager**  
handles making and receiving phone calls

22

# Android System Architecture

## Application Framework

- **Window Manager**

It is a system service, which is responsible for managing the z-ordered list of windows, which windows are visible, and how they are laid out on screen. Among other things, it automatically performs window transitions and animations when opening or closing an app or rotating the screen.

23

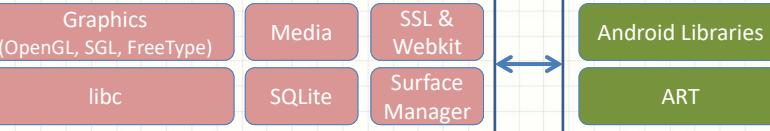
# Android System Architecture



## Application Framework



## Libraries



## Android Run Time



## Linux Kernel



24

## Android System Architecture

### Application Layer

- Apps can be written in **Java, Kotlin, Dart** or some other programming languages.
- We are using **Java**
- We write our apps to be installed on this layer only. Examples of applications are Games, Messages, Contacts etc.
- The application layer runs within the Android run time, using the classes and services made available from the application framework.

25

# Application Architecture Overview

26

## Android Versions

- Android has gone through a number of updates since its first release in 2008.

| Name               | Ver No.   | Initial stable release date | API level |
|--------------------|-----------|-----------------------------|-----------|
| Android 1.0 / 1.1  | 1.0 / 1,1 | Sep 23, 2008 / Feb 9, 2009  | 1 / 2     |
| Cupcake            | 1.5       | April 27, 2009              | 3         |
| Donut              | 1.6       | Sep 15, 2009                | 4         |
| Éclair             | 2.0~2.1   | Oct 27, 2009                | 5~7       |
| Froyo              | 2.2~2.2.3 | May 20, 2010                | 8         |
| Gingerbread        | 2.3~2.3.7 | Dec 6, 2010                 | 9~10      |
| Honeycomb          | 3.0~3.2.6 | Feb 22, 2011                | 11~13     |
| Ice Cream Sandwich | 4.0~4.0.4 | Oct 18, 2011                | 14~15     |
| Jelly Bean         | 4.1~4.3.1 | July 9, 2012                | 16~18     |
| KitKat             | 4.4~4.4.4 | Oct 31, 2013                | 19~20     |
| Lollipop           | 5.0~5.1.1 | Nov 4, 2014                 | 21~22     |
| Marshmallow        | 6.0~6.0.1 | Oct 2, 2015                 | 23        |
| Nougat             | 7.0~7.1.2 | Aug 22, 2016                | 24~25     |
| Oreo               | 8.0~8.1   | Aug 21, 2017                | 26~27     |
| Pie                | 9         | Aug 6, 2018                 | 28        |
| Quince tart        | 10        | Sep 3, 2019                 | 29        |
| Red Velvet Cake    | 11        | Se 8, 2020                  | 30        |
| Snow Cone          | 12        | Oct 4, 2021                 | 31~32     |
| Tiramisu           | 13        | Aug 15, 2022                | 33        |
| Upside Down Cake   | 14        | Q3 2023                     | 34        |
| Vanilla Ice Cream  | 15        | Q2 or Q3 2024               | 35        |

27

## Android Studio Versions vs AGP

| Android Studio Name | Ver No.         | Required Android Gradle plugin (AGP) version |
|---------------------|-----------------|--|
| Arctic Fox          | 2020.3.1        | 3.1 – 7.0                                    |
| Bumblebee           | 2021.1.1        | 3.2 – 7.1                                    |
| Chipmunk            | 2021.2.1        | 3.2 – 7.2                                    |
| Dolphin             | 2021.3.1        | 3.2 – 7.2                                    |
| Electric Eel        | 2022.1.1        | 3.2 – 7.4                                    |
| Flamingo            | 2022.2.1        | 3.2 – 8.0                                    |
| <b>Giraffe</b>      | <b>2022.3.1</b> | <b>3.2 – 8.1</b>                             |
| Hedgehog            | 2023.1.1        | 3.2 – 8.2                                    |
| Iguana              | 2023.2.1        | 3.2 – 8.3                                    |
| Jellyfish           | 2023.3.1        | 3.2 – 8.4                                    |
| Dolphin             | 2021.3.1        | 3.2 – 7.2                                    |

28

## API vs minimum

| Android API level | Min Android Studio version | Min AGP version | Min Gradle version | Min JDK version |
|-------------------|----------------------------|-----------------|--------------------|-----------------|
| 33                | Flamingo   2022.2.1        | 7.2             | 7.3.3              | 11              |
| 34                | Hedgehog   2023.1.1        | 8.1.1           | 8.0                | 17              |
| 35                | Jellyfish   2023.3.1       | 8.4             | 8.6                |                 |

29

## Android Application Architecture

- The architecture of an Android app **differs from desktop application** architecture
- App architecture is **based upon components that communicate with each other by using intents** that are described by a **manifest** and are stored in an app package
- **Application components** are the essential building blocks of an Android application

30

## Android Application Architecture

- Android apps **do not have a single entry point** (no C-style main() function, for example). Instead, **apps use components that are instantiated and run as needed**
- Each component exists as its own entity and plays a specific role - each one is a unique building block that helps define your application's overall behaviour.
- Each component is an entry point through which the system or a user can enter your app.

31

## Android Application Architecture

- There are **four different types of application components**
- Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

32

## Android Application Architecture

Here are the four types of application components:

- Activities
- Services
- Content Providers
- Broadcast Receivers

33

## Android Application Components

### Activities

- An **activity** is the entry point for interacting with the user. It represents a single screen with a user interface.
- For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.

34

# Android Application Components

## Activities

- Although the activities work together to form a cohesive user experience in the email app, **each one is independent of the others**.
- As such, a different app can start any one of these activities if the email app allows it. **For example, a camera app can start the activity in the email app that composes new mail to allow the user to share a picture.**
- An *activity* is implemented as a **subclass of Activity**

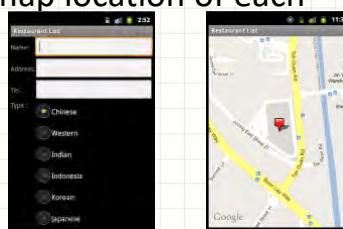
35

# Android Application Components

## Activities

- Example

A restaurant list application might have one activity that shows a list of restaurants, another activity to display the detail of each restaurant, and another activity that show the map location of each restaurant.



36

# Android Application Components

## Services

- A **service** is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.
- It is a **component that runs in the background** to perform long-running operations or to perform work for remote processes.
- A service **does not provide a user interface**.

37

# Android Application Components

## Services

- The service runs in the background indefinitely even if application is destroyed. It is stopped by **stopService()** method or can stop itself by calling the **stopSelf()** method.
- A service is implemented as a subclass of **Service**.
- Example
  - A service might play music in the background (i.e. even the app is ‘minimised’) while the user moves on to a different application, or it might fetch data over the network without blocking user interaction with an activity.

38

# Android Application Components

## Broadcast Receivers

- A **broadcast receiver** (*receiver*) is an Android component which allows you to register for **system** or **application** events.
- All registered receivers for an event are **notified by the Android runtime once this event happens**.
- **Broadcast Receivers** simply respond to the **registered** broadcast messages from other applications or from the system itself.

39

# Android Application Components

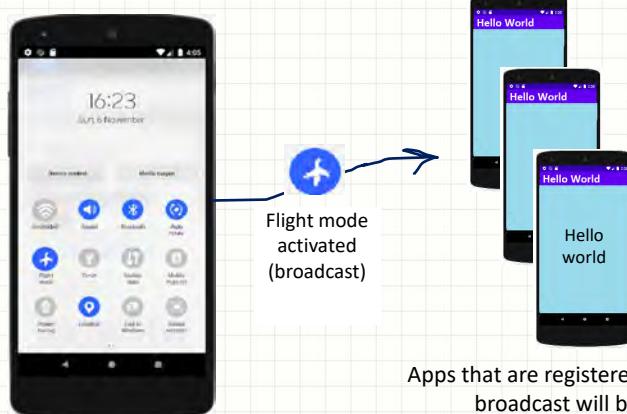
## Broadcast Receivers

Examples:

- An application can initiate broadcast to let other applications know that some data has been downloaded to the device and is available for them to use, so those registered broadcast receivers will intercept this communication and will initiate appropriate action.
- Applications can register for the **ACTION\_BOOT\_COMPLETED** **system event** which is fired once the Android system has completed the boot process.

40

## Example



Apps that are registered to receive such broadcast will be notified.  
The code their respective broadcast receiving method will run.

41

## Android Application Components

### Broadcast Receivers

- A *broadcast receiver* is implemented as a **subclass of BroadcastReceiver** and each broadcast is delivered as an **Intent** object.
- Many broadcasts **originate from the system**

#### Example

A broadcast announcing that the screen has turned off, the battery is low (`android.intent.action.BATTERY_LOW`) , or a picture was captured

42

# Android Application Components

## Broadcast Receivers

- Apps can also initiate broadcasts

### Example

An app may want to let other apps know that some data has finished downloading from the network to the device and is available now for usage

- Although broadcast receivers do not display a user interface, they may **create a status bar notification** to alert the user when a broadcast event occurs.

43

# Android Application Components

## Content Providers

- A **content provider** manages a shared set of **application data** which is stored in the file system, an SQLite database, on the web, or any other persistent storage location the app can access
- Through the content provider, other apps can query or even modify the data

44

# Android Application Components

## Content Providers

- Example

The Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as `ContactsContract.Data`) to read and write information about a particular person

- A content provider is implemented as a subclass of `ContentProvider` and must implement a standard set of APIs that enable other applications to perform transactions.

45

# Intent & IntentFilter

46

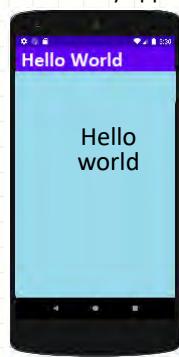
## Activating Components

- Three of the four component types (**activities, services, and broadcast receivers**) are activated by an asynchronous message called an *intent*
- Intents bind individual components to each other at runtime. You can think of them as the **messengers** that request an action from other components, whether the component belongs to your application or another.

47

## Example

1-activity app



A simple 1-page Hello World app

2-activity app

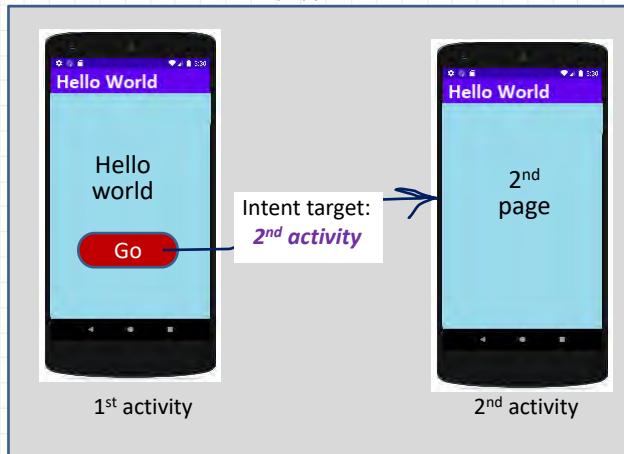


Then, there is a need to communicate from the 1<sup>st</sup> activity to 2<sup>nd</sup> activity. This is done via an *intent*.

48

## Example explicit intent

2-activity app

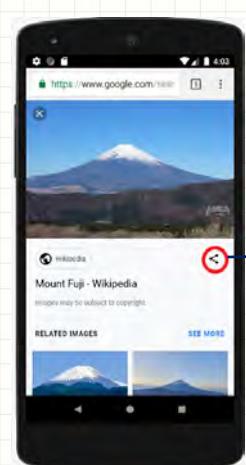


The intent used to communicate within the same app is usually **explicit intent**.

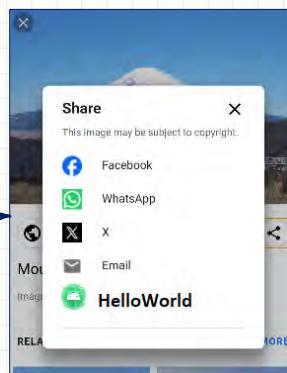
As it is coded clearly in the program that which the **targeted activity** is.

49

## Example – implicit intent



Intent target not specified in the program. The **target app** is chosen by the user at **runtime**.



The intent used to communicate for this sharing of images is done via **implicit intent**, to all app that have registered to be able to receive images.

50

## Activating Components

- *Intents* are asynchronous messages which allow application components to request functionality from other Android components.
- Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

51

## Intents

- Intent is basically a message that is passed between components (such as Activities, Services, Broadcast Receivers, and Content Providers)
- It is almost equivalent to parameters passed to API calls

52

## Intents

- The fundamental differences between API calls and intents' way of invoking components are:
  - API calls are **synchronous** while intent-based invocations are **asynchronous**
  - API calls are **compile time** binding while intent-based calls are **run-time binding**

53

## Intents

- An intent is created with an **Intent** object, which defines a message to activate either a **specific component (explicit intent)** or a **specific type of component (implicit intent)**.
- Example
  - An Activity can send an **explicit** intent to start another Activity in the same app.
  - An intent (**implicit**) might **convey a request** for an activity to show an image or to open a web page.

54

## Intents

- There are separate mechanisms for delivering intents to each type of component (such as Activities, Services or Broadcast Receivers) :
  - To start another activity
    - An **Intent object** is passed to **startActivity()** to launch an activity
    - or
    - startActivityForResult()** to launch an activity with return result
- Example
- ```
Intent myintent = new Intent(...);  
startActivity(myintent);
```

55

## Intents

- To start or bind to a service
  - An **Intent object** is passed to **startService()** to start a service or deliver new instructions to an ongoing service.
  - Similarly, an intent can be passed to **bindService()** to bind to a running service.

56

## Intents

- To initiate a broadcast

Intent objects passed to any of the broadcast methods (such as `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()`) are delivered to all registered broadcast receivers.

57

## Intents

Intents can be classified as:

- Explicit Intents
- Implicit Intents

58

## Explicit Intents

- Explicit Intents explicitly names the component which should be called by the Android system, by using the Java class as identifier
- It is made to work exactly like API calls

59

## Explicit Intents

- The following shows an explicit Intent to start the associated activity class
- Example – explicitly start a HelloWorld activity

```
Intent myintent = new Intent(this, HelloWorld.class);
startActivity(myintent);
```
- When run, this code snippet has the following consequences:
  - A new instance of HelloWorld is created
  - The instance is pushed on top of the current task's stack, which is the one the calling activity is in.
  - The activity is started and brought to the foreground.

60

## Explicit Intents

- Example – passing data between two activities  
you want explicitly call the activity B from activity A and pass to it an array of integers:

```
int intArray[] = {1,2,3,4};  
Intent in = new Intent(this, B.class);  
in.putExtra("my_array", intArray);  
startActivity(in);
```

To read the information in activity B (in onCreate() method) you should use the following code:

```
Bundle extras = getIntent().getExtras();  
int[] arrayInB = extras.getIntArray("my_array");
```

61

## Implicit Intents

- Implicit Intents **do not specify the Java class which should be called**
- For an implicit intent, you describes the type of action to perform and, optionally, the data upon which you would like to perform the action.
- The implicit intent allows the Android system to find a component on the device that can perform the action and start it.
- If there are multiple components that can perform the action described by the intent, the user selects which one to use.

62

## Implicit Intents

- The **Android system finds the best component for handling the intent**. This is done by comparing the intent received with the **intent filters** provided in the **manifest file** of other apps on the device.
- The comparison to Intent Filter is done with three elements of the intent object namely **action**, **data** and **category**

63

## Implicit Intents

- Example

The following tells the Android system to view a webpage. Typically the web browser is registered to this Intent i.e. the browser's manifest file has an intent filter to view webpage. Other components could also register themselves to this event, in which case the Android system will allow the user selects which one to use.

```
Intent intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("http://www.sp.edu.sg"));
```

64

## Intent Filters

- If an Intent is sent to the Android system, it will determine suitable applications for this Intent
- If several components have been registered for this type of Intent, Android offers the user's choice to open one of them
- This determination is based on IntentFilters. An IntentFilter **specifies the types of Intent that an activity, service, or broadcast receiver can respond**

65

## Intent Filters

- An **intent filter** is an expression in an app's manifest file that specifies the type of intents that the component would like to receive.
- For instance, by declaring an **intent filter** for an activity, you make it possible for other apps to directly start your activity with the **declared intent**.

66

## Intent Filters

- Hence, Intent Filters **describe the capability of the component** (Activity, Service or Broadcast Receivers) **to handle an implicit intent.**
- It specifies what an activity or service can do and what types of broadcasts a receiver can handle
- **Components without intent filters** cannot receive implicit intents, but can only handle **explicit intents**

67

## Intent Filters

- Intent filters are typically **defined in the "AndroidManifest.xml" manifest file.**
- For BroadcastReceiver it is also possible to define them in coding
- An **intent filter** is defined by its **category**, **action** and **data** filters. It can also contain additional metadata.

68

## Intent Filters

- Example

The following will register an Activity for the intent which is triggered when someone wants to open a webpage

```
<activity android:name=".BrowserActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http"/>
    </intent-filter>
</activity>
```

69

## Intent Filters

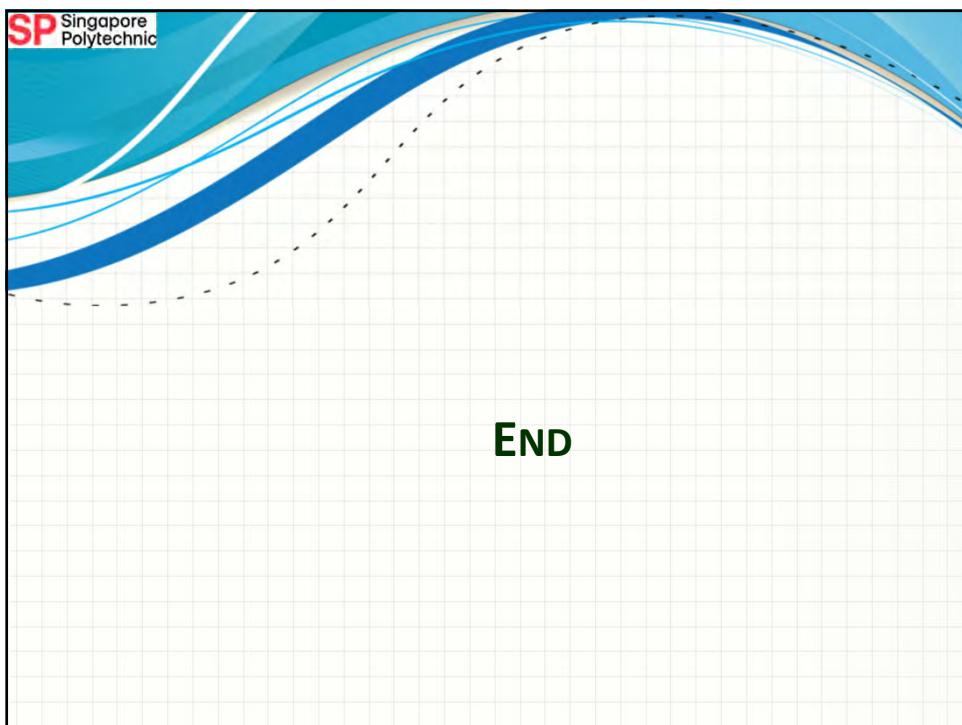
- Example

The following will register an activity for the ACTION.SEND intent for the “text/plain” mime type.

This activity can be started when an app tries to share text content by constructing an intent and passing it to startActivity(), your application will be listed as an option.

```
<activity android:name=".ActivityTest" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

70



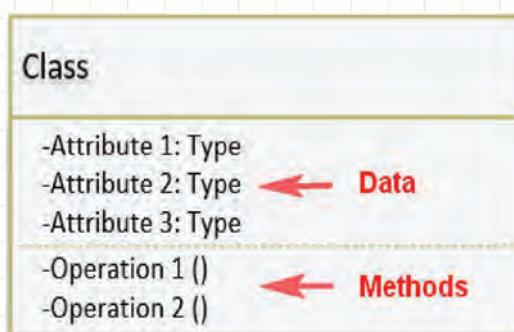
71

# Introduction to Object Oriented Programming (OOP)



## Object-Oriented Programming (OOP)

- Java is a **class-based** object-oriented programming (OOP) language that is built around the concept of “**objects**” that contain **data (properties or attributes)** and **methods (behaviours or functions)**.



- The primary purpose of object-oriented programming is to increase the **flexibility** and **Maintainability** of programs.



# Object-Oriented Programming (OOP)

- The primary purpose of object-oriented programming is to increase the **flexibility** and **maintainability** of programs.
- The main principles of object-oriented programming are:
  1. Data Abstraction
  2. Data Encapsulation
  3. Inheritance
  4. Polymorphism



# Object-Oriented Programming (OOP)

## • **Data Abstraction**

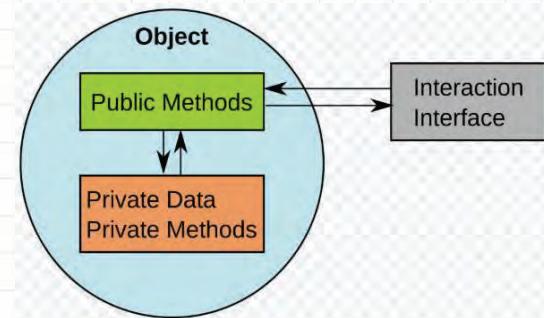
- Means providing only **essential information** to the user and hiding their **background details**.
- It is the process by which data and methods are defined with a representation that model real-world objects while hiding away the implementation details.
- For example, when you login to your Amazon account online, you enter your user\_id and password and press login, what happens when you press login, how the input data sent to amazon server, how it gets verified is all abstracted away from the you. Here prompting the user to enter user\_id and password and press login are presented to the user, but the implementation details of the how the user is validated are hidden.



# Object-Oriented Programming (OOP)

## • Encapsulation

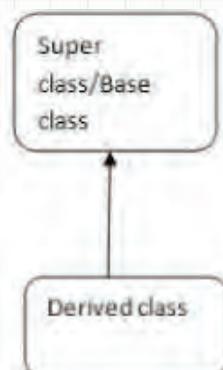
- It is **wrapping the data and methods** that manipulates it, together into a **single unit** along with **protecting** the data and methods from the outside world.
- It is actually the technique of making **data private** and providing access to those data via **public methods**.
- If a data field of a class is declared **private**, it cannot be accessed by anyone outside the class, thereby hiding the data fields within the class. For this reason, encapsulation is also referred to as **data hiding**.
- Java uses the keywords **private** and **public** to control the access to data fields and methods'



# Object-Oriented Programming (OOP)

## • Inheritance

- It is the process by which one class acquires the properties and functionalities of another class.
- It makes it possible to create a **child class** that **inherits** the **data** and **methods** of the **parent class**.
- The child class can **override** the data and methods of the parent class.
- The child class can also **add** new data and functionality besides those inherited from the parent class.
- Parent classes are also called **superclasses** or **base** classes, while child classes are known as **subclasses** or **derived** classes as well.
- Hence, inheritance is used to reuse an existing class to build a new class. The child inherits **visible (public)** properties and methods from its parent while adding additional properties and methods of its own.



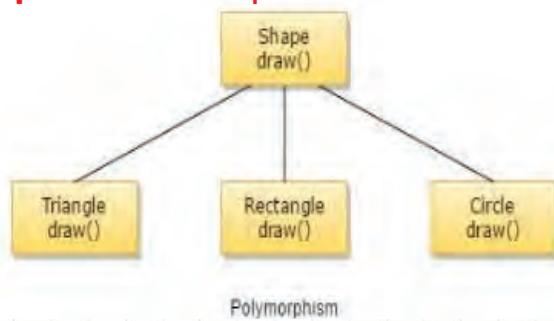
# Object-Oriented Programming (OOP)

## • Polymorphism

- It is the ability to perform a certain action in **different ways**. In other words it means, one method with multiple implementations, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation.
- In Java, polymorphism can take two forms: method **overloading** and method **overriding**.
- **Overloading** in simple words means more than one method having the **same method name** but **different type signature**. **Type signature** refer to the **number, order**, and **types** of the method parameters. When they are called they are differentiated by the number, order, and types of their parameters. This is called **static** because, which method to be invoked is decided at the time of compilation.
- **Overriding** means a derived class is implementing a method of its super class. When a method in a derived class has **same name** and **type signature** as a method in its super class, then the method is known as overridden method. The call to overridden method is resolved at runtime, thus called **runtime polymorphism**.
- For more info(you may visit it again after each Lab exercise):
  - <https://www.programiz.com/java-programming/polymorphism>

# Object-Oriented Programming (OOP)

- For example, we have a **base class Shape** that has a **public method draw()** that draw a shape image.



- By inheritance, we derived **Triangle**, **Rectangle** and **Circle** from the base class, **Shape**. The public method **draw()** of **Shape** is inherited by **Triangle**, **Rectangle** and **Circle**.
- Since the **inherited** method, **draw()** from **Shape**, draw only the shape image, **Triangle**, **Rectangle** and **Circle** each need to **override** its inherited **draw()** method so that it draw the triangle, rectangle and circle images respectively.
- Hence the method **draw()** has different implementations i.e. **polymorphism**.

# Objects and Classes – **class** keyword

- Writing object-oriented programs involves
  - creating classes
  - creating objects from those classes
  - creating applications, which are stand-alone executable programs that use those objects.
- A class is a **template** or a **blueprint** from which objects are created. It contains **data fields (attributes)** and **methods** (behaviours).
- A class declaration is **normally a single file** in a Java project, in which you declare the **data fields** and **methods** that objects of the class will provide.
- An **object** is an **instance of a class**. You can create **many instances of a class**.
- Objects of the same class have the same data fields and methods except that each object has different data **field values**.

Syntax

```
public class <class_name> {  
    data fields;  
    methods;  
}
```



# Objects and Classes – **class** keyword

- To create a class, use the keyword **class**. For example:
  - Here "**name**" is a data field of the **String** data type. It has **private** access (data encapsulation) and can only be accessed within the same class (an outside class have no access to it).
  - However, it is possible to access them if we provide **public getter** and **setter** methods.
  - **getName()** is a public getter method that returns the value of the variable "**name**"
  - **setName()** is a public setter method that takes a parameter (**newName**) and assigns it to the "**name**" variable. The **this** keyword is used to refer to the current object.

```
public class Person {  
    private String name; // private = restricted access  
  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```

Object's variable name's value

# Objects and Classes – new keyword

- As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the **new** keyword is used to create new objects.

Syntax:

```
class_name objectname = new class_name();
```

- For example using the **Person** class we create two objects, **personObj1** and **personObj2**

```
Person personObj1 = new Person();
```

```
Person personObj2 = new Person();
```



# Objects and Classes – dot(.) operator

- To access the data field and method of an object, you use the dot notation using a dot (.) operator.

Syntax:

```
objectname.datafield;  
objectname.method;
```

- For example

```
Person personObj1 = new Person();
```

```
personObj1.name = "Simon"; // error because name is private
```

- However, as the **name** variable of class **Person** is declared as **private** we **cannot** access it from outside this class. Hence **personObj1.name** will result in syntax error.

```
/Person.java:18: error: name has private access in Person  
    personObj1.name = "Simon";  
               ^  
  
/Person.java:19: error: name has private access in Person  
    System.out.println(personObj1.name);  
               ^
```



# Objects and Classes – dot(.) operator

- Instead, we use the public `getName()` and `setName()` methods to access and update the variable, `name`.

- To update the variable, `name` to "Simon":

```
personObj1.setName("Simon"); //update name to "Simon"
```

- To access the variable, `name`:

```
personObj1.getName(); //return the content of name.
```

- Note that each **Java class** need to be saved as a **single file** with the file extension `.java` and the **filename must exactly be the same as the class name** i.e. **case-sensitive**. Hence class **Person** will be saved as **Person.java**. It will be an **error** if the filename is **person.java**.



# Objects and Classes – private and public keywords

- Example:

```
1  public class Person {  
2      private String name;  
3  
4      // Getter  
5      public String getName() {  
6          return name;  
7      }  
8  
9      // Setter  
10     public void setName(String newName) {  
11         this.name = newName;  
12     }  
13 }  
14  
15 public class Main {  
16     public static void main(String[] args) {  
17         Person personObj1 = new Person();  
18         personObj1.name = "Simon";  
19         System.out.println(personObj1.name);  
20     }  
21 }  
22  
/Person.java:18: error: name has private access in Person  
    personObj1.name = "Simon";  
                           ^  
/Person.java:19: error: name has private access in Person  
    System.out.println(personObj1.name);  
                           ^
```

**public** keyword declares a member's access as **public**.

- public members are visible to all other classes.
- This means that any other class can access a public field or method.

**private** keyword declares a member's access as **private**.

- That is, the member is only visible within the class, not from any other class (including subclasses).



# Objects and Classes – private and public keywords

- Example:

```
1  public class Person {  
2      private String name;  
3  
4      // Getter  
5      public String getName() {  
6          return name;  
7      }  
8  
9      // Setter  
10     public void setName(String newName) {  
11         this.name = newName;  
12     }  
13 }  
14  
15 public class Main {  
16     public static void main(String[] args) {  
17         Person personObj1 = new Person();  
18         personObj1.setName("Simon");  
19         System.out.println(personObj1.getName());  
20     }  
21 }  
22 }
```

Simon



## Constructor

- A constructor in Java is a **special method** that is used to initialize objects.
- It is called when an object of a class is created.
- It can be used to set initial values for object attributes (data fields)
- Note that the constructor name must **match the class name**, and it **cannot have a return type** not even the **void** type.
- Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a **default constructor** for that class.



# Constructor

- In this example we have two constructors that have the **same method name** but **different type signature** i.e. `Person()` and `Person(String newName)`. This is known as **method overloading**.

```
public class Person {  
    private String name;  
    // constructor  
    public Person() {  
        this.name = "";  
    }  
    // constructor with input parameter  
    public Person(String newName) {  
        this.name = newName;  
    }  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        //call the constructor Person()  
        // to initialize name of object personObj1 to a blank string  
        Person personObj1 = new Person();  
        //call the constructor Person(String newName)  
        //to initialize name of object personObj2 to "John"  
        Person personObj2 = new Person("John");  
        //update name of object personObj1 to "Simon"  
        personObj1.setName("Simon");  
        //Output name of object personObj1  
        System.out.println(personObj1.getName());  
        //Output name of object personObj2  
        System.out.println(personObj2.getName());  
    }  
}
```

Simon  
John



## It's Quiz Time!

- Assume Person is a created class, for statement below:  
`Person tom = new Person("Tom");`
  - The class name is \_\_\_\_\_.
  - The object name is \_\_\_\_\_.
  - The constructor name is \_\_\_\_\_.
  - How many parameter does the constructor take?  
\_\_\_\_\_.
  - What's the purpose of constructor?  
\_\_\_\_\_.

More Java OOP resource:

[https://www.tutorialspoint.com/java/java\\_object\\_classes.htm](https://www.tutorialspoint.com/java/java_object_classes.htm)



# Interface

- A Java interface is a bit like a class, except a Java interface can only contain method signatures and data fields.
- A Java interface cannot contain an implementation of the methods, only the signature (name, parameters and exceptions) of the method.
- A class that has at least one method that has no implementation, is known an abstract class. Therefore, an interface is an abstract class.
- You can use interfaces in Java as a way to achieve polymorphism.
- Example:

```
// interface  
public interface Animal {  
    public void animalSound(); // interface method (does not have a body)  
    public void run(); // interface method (does not have a body)  
}
```



## Inheritance And Polymorphism – extends keyword

- Inheritance can be defined as the process where one class inherit attributes (data fields) and methods from another.
- The class that inherits from another class is known as subclass (derived class, child class) and the class being inherited from is known as superclass (base class, parent class).
- To inherit from a class, use the extends keyword.
- A class can only extend from one parent class.
- Object class:  
<https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Object.html>

### Syntax:

```
class Superclass {  
    ....  
}  
class Subclass extends Super {  
    ....  
}
```



# Inheritance And Polymorphism

## – super keyword

- The **super** keyword in Java is a reference variable which is used to refer **immediate** parent class object.
- Whenever you create an instance of the subclass, an instance of the parent class is created implicitly which is referred by the **super** reference variable.
- Usage of Java **super** keyword:
  - We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{  
    String color="white";  
}  
  
class Dog extends Animal{  
    String color="black";  
    void printColor(){  
        System.out.println(color); //prints color of Dog class i.e. black  
        System.out.println(super.color); //prints color of Animal class i.e. white  
    }  
}
```

In the above example, both Animal and Dog classes have a **common attribute color**. If we print the color attribute, it will print the color of current class by default. To **access the parent attribute**, we need to use **super** keyword.

# Inheritance And Polymorphism

## – super keyword

- The super keyword can also be used to **invoke parent class method**. It should be used if subclass contains the same method as the parent class. In other words, it is used if method is **overridden**.

```
class Animal{  
    void eat(){  
        System.out.println("eating...");  
    }  
}  
class Dog extends Animal{  
    void eat() { //override method  
        System.out.println("eating bread...");  
    }  
    void bark() {  
        System.out.println("barking...");  
    }  
    void work(){  
        super.eat(); //call the eat() method of the Animal class  
        eat(); //call the eat() method of the Dog class  
        bark(); //call the bark() method of the Dog class  
    }  
}
```

In this example, both Animal and Dog classes have **eat()** method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local method. To call the parent class method, we need to use the **super** keyword.

# Inheritance And Polymorphism

## – super keyword

- iii. The super keyword can also be used to invoke the parent class constructor.

```
class Animal {  
    Animal() {      //default constructor  
        System.out.println("animal is created");  
    }  
}  
  
class Dog extends Animal {  
    Dog() {          //default constructor  
        super();    //call default constructor Animal() of Animal class  
        System.out.println("dog is created");  
    }  
}
```

- In the above example, the Dog class constructor invoke the parent Animal class constructor using the **super()** method.
- You can **only invoke super()** in a constructor method.
- Call to super() must be the **first statement** in derived class constructor.
- “super” can call both **parametric (overloaded constructor)** as well as **non-parametric (default constructor)** parent constructors depending upon the situation.

# Inheritance And Polymorphism

## – super keyword

```
public class Person {  
    int id;  
    String name;  
    Person(int id, String name) {  
        this.id=id;  
        this.name=name;  
    }  
    void display() {      // override method  
        System.out.println("ID : " + id);  
        System.out.println("Name : " + name);  
    }  
}
```

```
public class Emp extends Person {  
    int salary;  
    Emp(int id, String name, int salary) {  
        super(id, name);    //invoking parent constructor with parameters  
        this.salary=salary;  
    }  
    void display() {      // override method  
        super.display();   //invoke parent display() method  
        System.out.println("Salary : $" + salary);  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Emp staff = new Emp(1,"Simon",5000);  
        staff.display();  
    }  
}
```

**Output:**  
ID : 1  
Name : Simon  
Salary : \$5000

# Inheritance And Polymorphism – implement keyword

- A Java **interface** is a bit like a class, except a Java interface can only contain method signatures and fields. An Java interface cannot contain an implementation of the methods, only the signature (name, parameters and exceptions) of the method.
- Before you can really use an interface, you must implement that interface in some Java class using the **implement** keyword.
- A class that implements an interface **must implement all the methods declared in the interface**. The methods must have the **exact same signature** (name + parameters) as declared in the interface.
- The class does **not need to implement (declare)** the variables (data fields) of an interface. Only the methods.
- Once a Java class implements an Java interface you can use an instance of that class as an instance of that interface.



# Inheritance And Polymorphism – implement keyword

```
// interface
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
    }
}

class MyMainClass {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

The pig says: wee wee  
Zzz



# Inheritance And Polymorphism – implement keyword

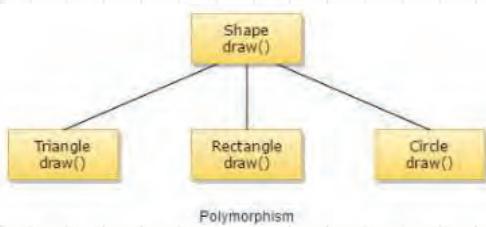
- A Java class can implement multiple Java interfaces. In that case the class must implement all the methods declared in all the interfaces implemented.
- You list the names of the interfaces to implement after the implements keyword, separated by a comma.

```
public interface MyInterface {  
    public void sayHello();  
}  
  
public interface MyOtherInterface {  
    public void sayGoodbye();  
}  
  
public class MyInterfaceImpl implements MyInterface, MyOtherInterface {  
  
    public void sayHello() {  
        System.out.println("Hello");  
    }  
  
    public void sayGoodbye() {  
        System.out.println("Goodbye");  
    }  
}
```



## Inheritance And Polymorphism - Example

- Polymorphism Example



```
public interface Shape {  
    public void draw(); // abstract method  
}  
  
public class Triangle implements Shape {  
    public void draw(){ // override method  
        System.out.println("Drawing triangle");  
    }  
}  
  
public class Rectangle implements Shape{  
    public void draw(){ // override method  
        System.out.println("Drawing rectangle");  
    }  
}  
  
public class Circle implements Shape{  
    public void draw(){ // override method  
        System.out.println("Drawing circle");  
    }  
}
```



# Inheritance And Polymorphism - Example

```
public class Main {  
    public static void main(String[] args) {  
        Shape sh = new Triangle(); // base class object, sh reference a subclass Triangle object  
        sh.draw(); // polymorphism - call the draw() of class Triangle  
        sh = new Rectangle(); // base class object, sh reference a subclass Rectangle object  
        sh.draw(); // polymorphism - call the draw() of class Rectangle  
        sh = new Circle(); // base class object, sh reference a subclass Circle object  
        sh.draw(); // polymorphism - call the draw() of class Circle  
    }  
}
```

Output:

Drawing triangle  
Drawing rectangle  
Drawing circle

- A reference variable of the base class can reference any object of its subclass. Here **sh** is reference variable of **Shape**.
- The variable **sh** can reference **object of subclass** Triangle, Rectangle and Circle.
- When we invoke the override method **draw()** through **sh**, at runtime the correct **draw()** method is called **depending what object is sh referencing**.
- Hence when **sh** is referencing an object of class Circle, then the **draw()** method of Circle will be invoked, similarly when **sh** reference an Triangle object, **draw()** method of Triangle will be invoked and when **sh** reference an Rectangle object, **draw()** method of Rectangle will be invoked.
- This multiple behaviours of **draw()** polymorphism.
- through the base class reference variable, **sh**, is known as



## It's Quiz Time!

- Assume Triangle and Drawable is properly designed, given following code segment:

```
public class RightTriangle extends Triangle implements  
Drawable
```

- Parent class name is \_\_\_\_\_.
- Child class name is \_\_\_\_\_.
- Interface name is \_\_\_\_\_.

More Java OOP resource:  
<http://pub.bruckner.cz/titles/oop>



# Object-Oriented Programming –

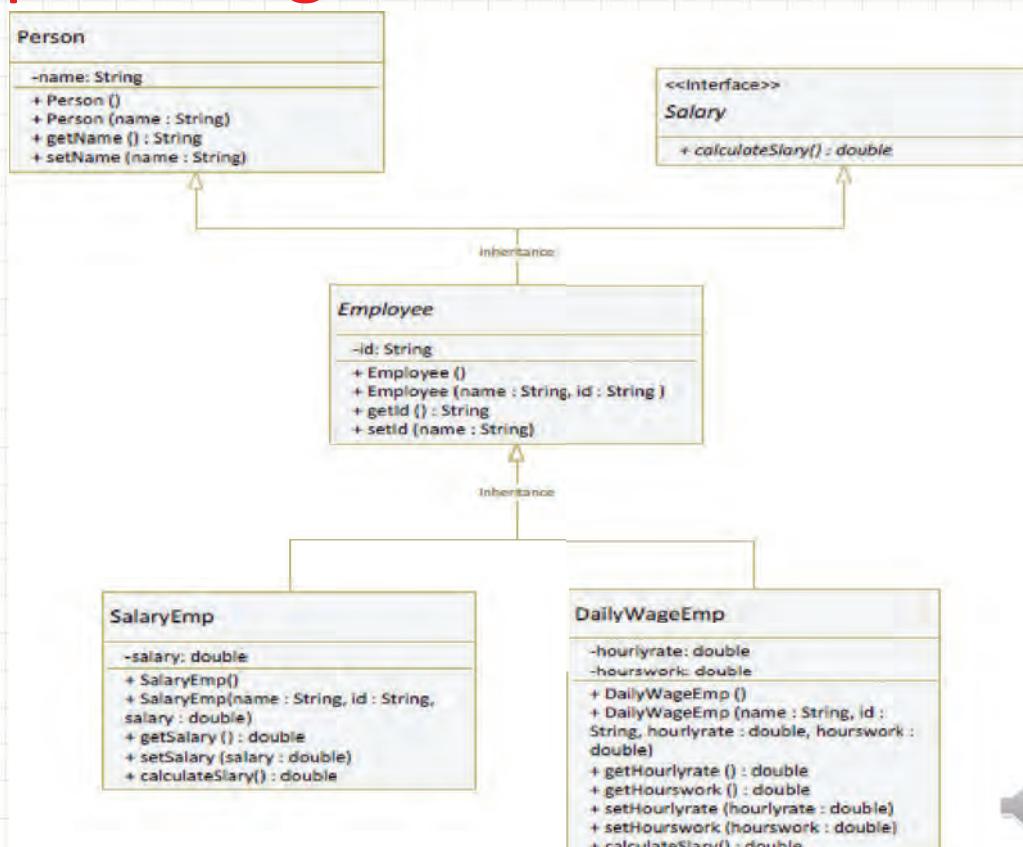
## Example Using Android Studio

- We will now program an OOP example using Android Studio using the class diagram below.
- Note that (+) means **public** access and (-) **private** access.
- Class **Employee** extends from class **Person** and implement **interface Salary**. It inherit the public methods from Person and Salary. It has added a new attribute, **id**.
- As class **Employee** inherit the abstract method **calculateSalary()** from interface **Salary** and does not provide an implementation for the method, it need to be declared as an **abstract** class.
- Classes **SalaryEmp** and **DailyWageEmp** extend from class **Employee**. Each class inherit all the public methods from class **Employee**.
- Class **SalaryEmp** has added a new attribute, **salary**.
- Class **DailyWageEmp** has added two new attributes, **hourlyrate** and **hourswork**.
- Classes **SalaryEmp** and **DailyWageEmp** have each **override** the abstract method **calculateSalary()** and provide a class specific implementation.



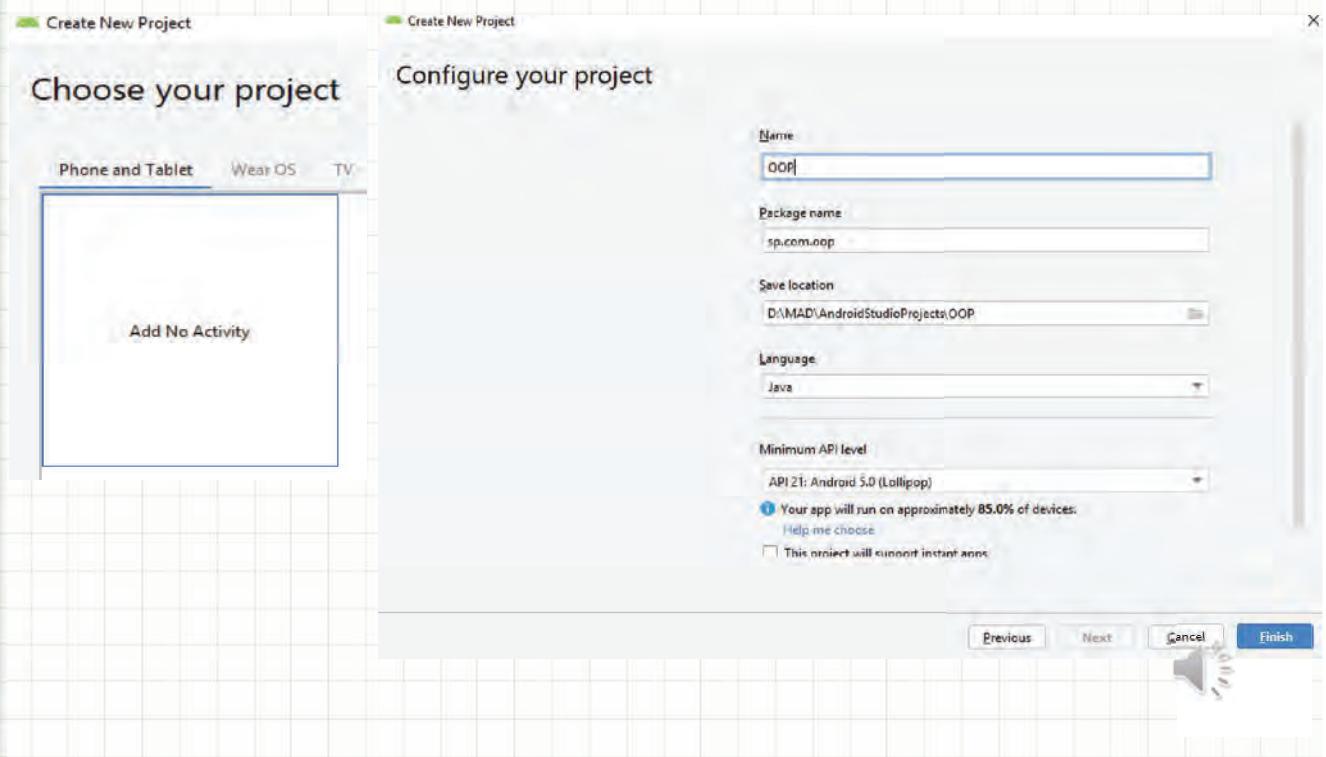
# Object-Oriented Programming –

## Example Using Android Studio



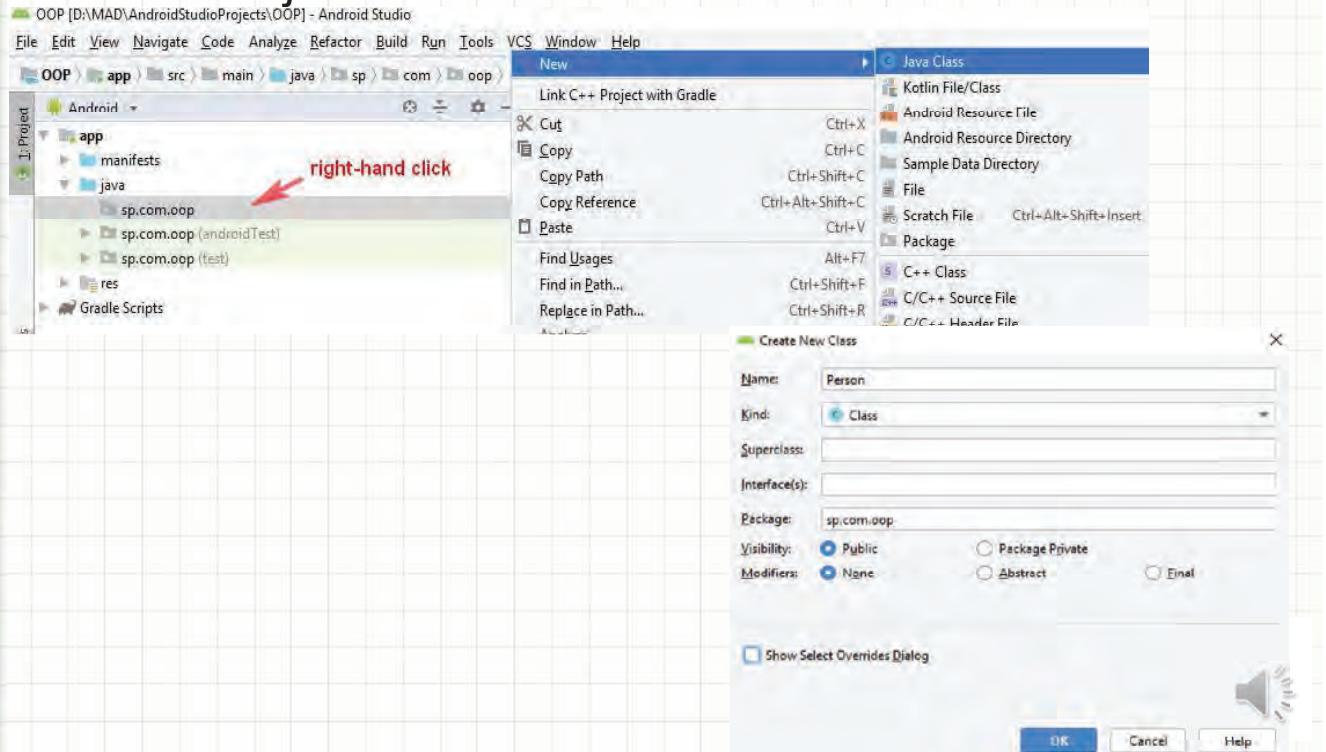
# Object-Oriented Programming – Example Using Android Studio

- Create new empty project in Android Studio.



# Object-Oriented Programming – Example Using Android Studio

- Create new java class Person.



# Object-Oriented Programming –

## Example Using Android Studio

- Type in the codes for class Person

The screenshot shows the Android Studio interface. On the left is the Project Navigational Drawer, which lists the project structure: app (manifests, java, sp.com.oop), sp.com.oop (Person), sp.com.oop (androidTest), sp.com.oop (test), res, and Gradle Scripts. The Person.java file is currently selected. On the right is the main editor window titled "Person.java". It contains the following Java code:

```
1 package sp.com.oop;
2
3 public class Person {
4     private String name;
5     public Person() { //default constructor
6         name = "";
7     }
8     public Person(String name) { //constructor
9         this.name = name;
10    }
11
12    public String getName() { //getter for name
13        return name;
14    }
15
16    public void setName(String name) { //setter for name
17        this.name = name;
18    }
19 }
```

A small speaker icon in the bottom right corner indicates that there is audio content associated with this slide.

# Object-Oriented Programming –

## Example Using Android Studio

- Create an interface Salary.
- Type in the codes for interface Salary.

The screenshot shows the "Create New Class" dialog box on the left and the main editor window on the right. The dialog box has the following settings:

- Name: Salary
- Kind: Interface
- Interface(s):
- Package: sp.com.oop
- Visibility: Public

The "OK" button at the bottom of the dialog is highlighted. In the main editor window, the Person.java file is open, and the following Java code is visible:

```
1 package sp.com.oop;
2
3 public interface Salary {
4     public double calculateSalary();
5 }
```

If you can't see the above dialog, you  
may manually type interface declaration.

# Object-Oriented Programming –

## Example Using Android Studio

- Create a Java class **Employee** that extends class **Person** and implement interface **Salary**. Ensure that “**Abstract**” option is checked.
- Since we are not implementing the interface method, `calculateSalary()`, the class **Employee** need to be **abstract**.
- Type in the codes for class **Employee**.

The screenshot shows the "Create New Class" dialog box and the code editor for the `Employee.java` file. The dialog box has the following settings:

- Name: Employee
- Kind: Class
- Superclass: sp.com.oop.Person
- Interface(s): sp.com.oop.Salary
- Package: sp.com.oop
- Visibility: Public
- Modifiers: Abstract (selected)

The code editor displays the generated abstract class code:

```
package sp.com.oop;

public abstract class Employee extends Person implements Salary {
    private String id;
    public Employee() { //default constructor
        super();
        id = "";
    }
    public Employee(String name, String id) { //constructor
        super(name);
        this.id = id;
    }
    public String getId() { //getter for id
        return id;
    }
    public void setId(String id) { //setter for id
        this.id = id;
    }
}
```

If you can't see the above dialog, you may manually type class declaration.

# Object-Oriented Programming –

## Example Using Android Studio

- Create a Java class **SalaryEmp** that extends from class **Employee**.
- Type in the codes for class **SalaryEmp**.

The screenshot shows the "Create New Class" dialog box and the code editor for the `SalaryEmp.java` file. The dialog box has the following settings:

- Name: SalaryEmp
- Kind: Class
- Superclass: sp.com.oop.Employee
- Interface(s):
- Package: sp.com.oop
- Visibility: Public
- Modifiers: None

The code editor displays the generated class code:

```
package sp.com.oop;

public class SalaryEmp extends Employee {
    private double salary;
    SalaryEmp() { //default constructor
        super();
        this.salary = 0.0;
    }
    //invoke constructor Employee(String name, String id)
    SalaryEmp(String name, String id, double salary) {
        super(name, id);
        this.salary = salary;
    }
    public double getSalary() { //getter for salary
        return salary;
    }
    public void setSalary(double salary) { //setter for salary
        this.salary = salary;
    }
    @Override
    public double calculateSalary() { //implementation of calculateSalary()
        // since salary is fixed, no computation required
        return salary;
    }
}
```

If you can't see the above dialog, you may manually type class declaration.

# Object-Oriented Programming – Example Using Android Studio

- Create a Java class **DailyWageEmp** that extends from class **Employee**.
- Type in the codes for class **DailyWageEmp** – to be continued

The screenshot shows the 'Create New Class' dialog box on the left and the code editor on the right. The dialog box has the following settings:

- Name: DailyWageEmp
- Kind: Class
- Superclass: sp.com.oop.Employee
- Interface(s):
- Package: sp.com.oop
- Visibility: Public
- Modifiers: None

The code editor shows the generated Java code for the DailyWageEmp class, which extends Employee and overrides calculateSalary().

```
package sp.com.oop;

public class DailyWageEmp extends Employee {
    private double hourlyrate;
    private double hourswork;
    DailyWageEmp() { //default constructor
        super();
        this.hourlyrate = 0.0;
        this.hourswork = 0.0;
    }
    //invoke constructor Employee(String name, String id)
    DailyWageEmp(String name, String id, double hourlyrate, double hourswork) {
        super(name, id);
        this.hourlyrate = hourlyrate;
        this.hourswork = hourswork;
    }
    public double getHourlyrate() {
        return hourlyrate;
    }
    public double getHourswork() {
        return hourswork;
    }
}
```

If you can't see the above dialog, you may manually type class declaration

# Object-Oriented Programming – Example Using Android Studio

- Type in the rest codes for class **DailyWageEmp** -- continued

The screenshot shows the code editor on the left and the 'Create New Class' dialog box on the right. The code editor contains the remaining part of the DailyWageEmp class, including setHourlyrate(), setHourswork(), and calculateSalary().

```
    public void setHourlyrate(double hourlyrate) {
        this.hourlyrate = hourlyrate;
    }

    public void setHourswork(double hourswork) {
        this.hourswork = hourswork;
    }

    @Override
    public double calculateSalary() { //implementation of calculateSalary()
        double salary;
        if (hourswork > 40) {
            salary = hourlyrate * (40 + 1.5 * (hourswork - 40));
        } else {
            salary = hourlyrate * hourswork;
        }
        return salary;
    }
}
```

The 'Create New Class' dialog box has the following settings:

- Name: Main
- Kind: Class
- Superclass:
- Interface(s):
- Package: sp.com.oop
- Visibility: Public
- Modifiers: None

# Object-Oriented Programming – Example Using Android Studio

- Create a Java class **Main**.
- Type in the codes for class **Main**.

The screenshot shows the Android Studio interface. On the left, there's a file tree with files like Person.java, Employee.java, Salary.java, DailyWageEmp.java, and SalaryEmp.java. The Main.java file is open in the editor. The code defines a Main class with a main method that creates a SalaryEmp object named Simon, sets its name and ID, and calculates its salary. It then creates a DailyWageEmp object named John, sets his name and ID, and calculates his salary. Finally, it prints out the names and earn amounts of both objects. To the right of the editor is a 'Create New Class' dialog box. It has fields for Name (Main), Kind (Class selected), Package (sp.com.oop), Visibility (Public selected), and Modifiers (None). There are also checkboxes for Superclass, Interface(s), and Show Select Overrides Dialog. At the bottom are OK, Cancel, and Help buttons.

```

package sp.com.oop;

public class Main {
    public static void main(String[] args){
        SalaryEmp simon = new SalaryEmp(); //invoke default constructor
        simon.setName("Simon"); //invoke inherited method
        simon.setId("12345"); //invoke inherited method
        simon.setSalary(3000); //invoke own method
        //invoke parameterized constructor to initialize object
        DailyWageEmp john = new DailyWageEmp(name: "John", id: "12155", hourlyrate: 5, hourwork: 45);
        //polymorphism
        Person per = simon;
        System.out.println("Person name:"+ per.getName()); //per behaves like child simon
        System.out.println(simon.getName()+" earn $" +simon.calculateSalary());
        per = john;
        System.out.println("Person name:"+ per.getName()); //per behaves like child john
        System.out.println(john.getName()+" earn $" +john.calculateSalary());
    }
}

```

**Running Result:**

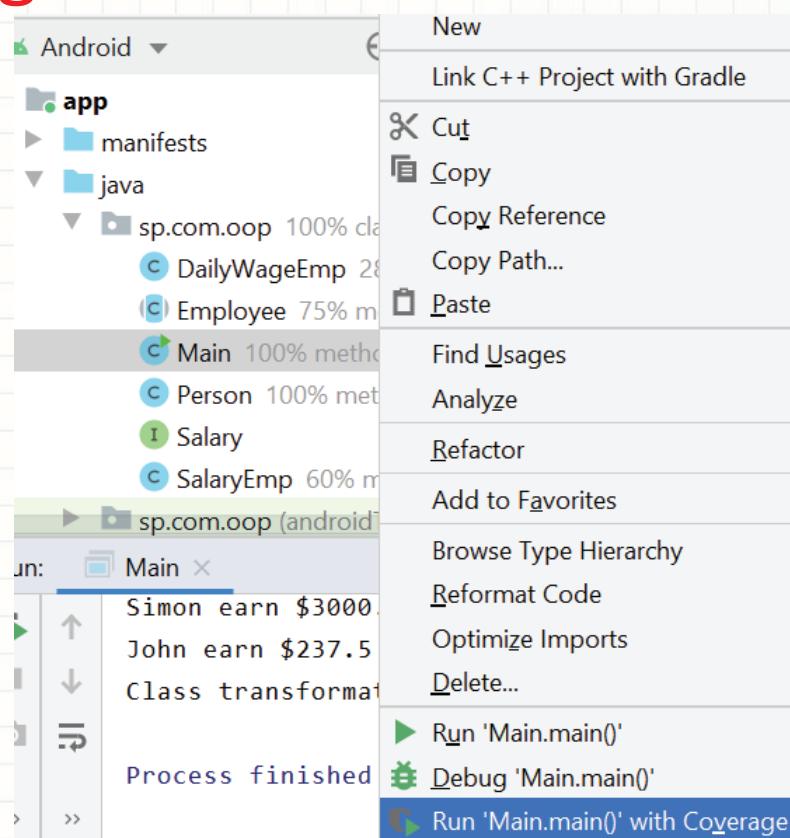
```

Person name:Simon
Simon earn $3000.0
Person name:John
John earn $237.5

```

# Object-Oriented Programming – Example Using Android Studio

- Click Hammer to build the project.
- Correct mistake, if any
- Run **Main**
  - Right click Java file **Main**
  - Click Run 'Main.main()' with Coverage



# OOP – package and import keywords

- A package in Java is used to group related classes.
- Packages are divided into two categories:
  - Built-in Packages (packages from the Java API)
  - User-defined Packages (create your own packages)
- To use a class or a package from the library, you need to use the **import** keyword.

## Syntax

```
import package.name.Class; // Import a single class  
import package.name.*; // Import the whole package
```

- If you find a class you want to use, for example, the Math class, which contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions, write the following code:

```
import java.lang.Math;
```

- To import a whole package, end the sentence with an asterisk sign (\*). The following example will import ALL the classes in the **java.util** package:

```
import java.util.*;
```



## It's Quiz Time!—Do it after you complete the OOP example

- Refer to slide 1 and 2 for the relationship among the 5 Java files (4 classes, 1 interface), and the Java application Main.java:
  - simon is an object created from class \_\_\_\_\_.
  - john has data type \_\_\_\_\_.
  - In statement simon.setId("12345");  
the setId method is inherited from class \_\_\_\_\_.
  - In statement simon.setSalary(3000);  
the setSalary method is from class \_\_\_\_\_.



# Getting Started



Official (Closed), Non-Sensitive

## Today's Overview

1

- Android Studio

2

- Android Project Structure

3

- Android Application Lifecycle

4

- Android UI

# Android Studio

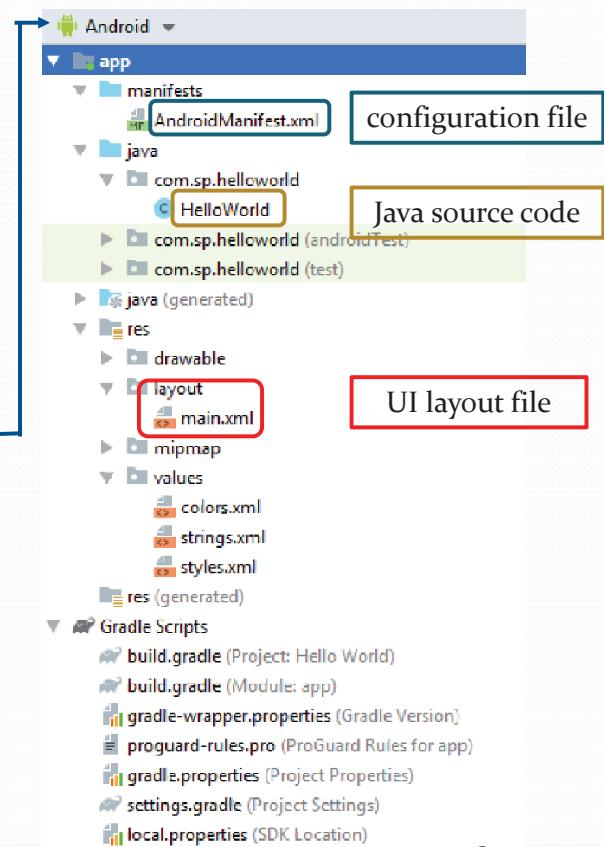
- Android Studio is the official IDE (integrated development environment) for developing Android Apps by Google.
- It is based on JetBrains' IntelliJ IDEA software and has lots of amazing features which helps developer in creating Android App.
- Refer to the lab “Getting Started with Android Studio” to
  - 1) Install the IDE software
  - 2) Install the Android SDK
  - 3) Configure the Android Virtual Device (AVD)
  - 4) Create and deploy the “HelloWorld” project to the emulator

## Understanding Hello World

# Android Project Structure

## Android Project Structure

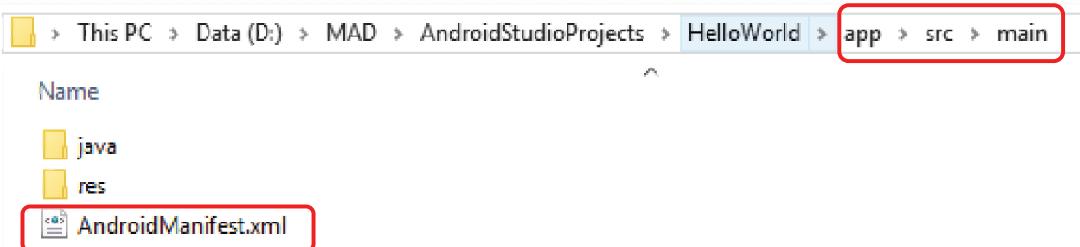
- When an android application, is created, the project structure will look similar as shown.
- This is the project structure of the HelloWorld created in the “Get Started with Android Studio” when choose “Android”



# AndroidManifest XML File

## AndroidManifest.xml

- Every app project must have an AndroidManifest.xml file.
- It resides in **app/src/main/**.



- The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

# AndroidManifest XML File

## AndroidManifest.xml

- We can declare following in AndroidManifest.xml:
- Package Name

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp"
    android:versionCode="1"
    android:versionName="1.0" >

    </manifest>
```

- Declaration of Activity, Services, Broadcast Receivers and Content Providers.

```
<manifest package="com.example.myapp" ... >
    <application ... >
        <activity android:name=".MainActivity" ... >
            ...
            </activity>
        </application>
    </manifest>
```

# AndroidManifest XML File

## AndroidManifest.xml

- Activity, Service and Broadcast Receiver may contains intent filters.

```
<manifest package="com.example.myapp" ... >
    <application ... >
        <activity android:name=".MainActivity" ... >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Permissions of Application.

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

# AndroidManifest XML File

- HelloWorld AndroidManifest.xml

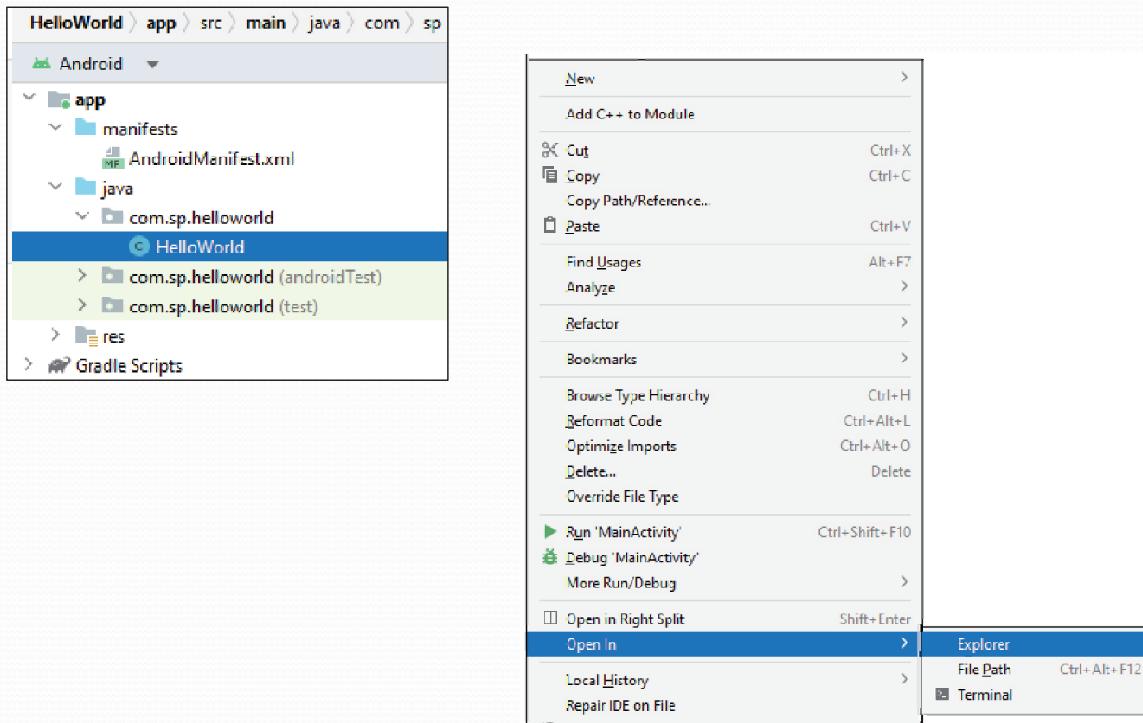
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Hello World"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.HelloWorld"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

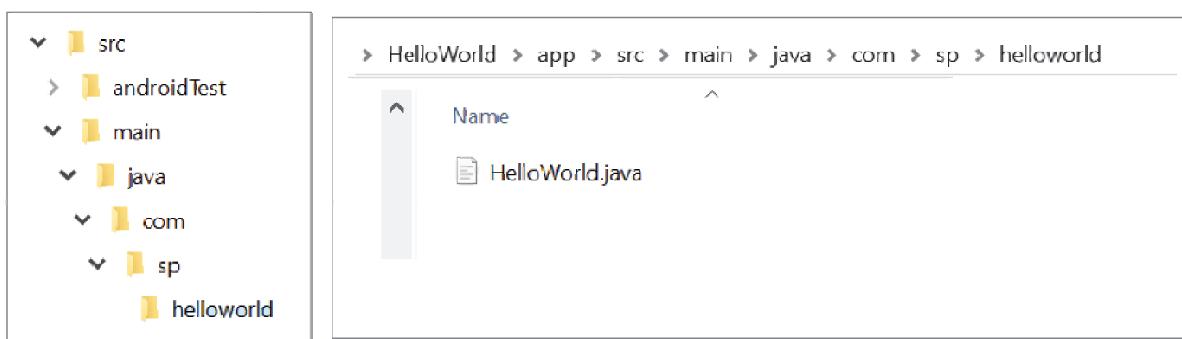
# Project Folder Structure

- Locating your java Folder: Right-click → Open in → Explorer



# Project Folder Structure

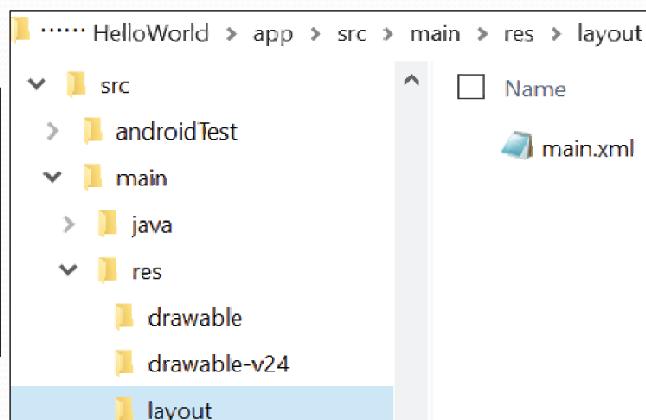
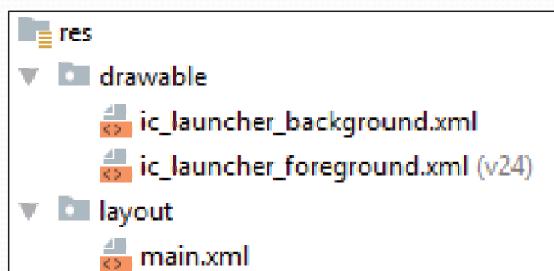
- java Folder:
  - Contains app Java classes source code in the **src/main/java** folder.  
e.g. `HelloWorld.java` is stored in under `java` folder in deeper folders corresponding to the app's package.  
Since `HelloWorld` is in the package “`com.sp.helloworld`”, it is under the subfolders `com/sp/helloworld`



# Project Folder Structure

- res Folder:

- **res(src/main/res)** is the folder which represents resource folder in android. It has multiple sub folders.
  - **drawable**: It is the folder in which you put images that you are going to use in your android app.
  - **layout**: It is the folder in which you have the layout xml file. Layout file represents design of the screen. Every activity class has a layout file corresponds to it. If you notice in HelloWorld.java , we have the “main.xml” layout file.



# Project Folder Structure

- res Folder:

**mipmap:**

- This folder contains the Android app icon in different size.

**values:**

- This folder will contains xml file which will have constant in it. For example:

**string.xml** file have all strings declared in it, so you don't have to hardcode anything in your java code.

**Why you should not hardcoded: ease of internationalization**

If you have all strings in English as of now and you are going to release your application in Chinese then you need to search all the strings and change them to Chinese in java code. But if you do not hardcode anything, you just have to convert strings.xml strings in Chinese and it will work.

# Project Folder Structure

- **Gradle Scripts Folder:**

- The Android build system compiles app resources and source code, and packages them into **APKs** that you can test, deploy, sign, and distribute.
- Android Studio uses **Gradle**, an advanced build toolkit, to automate and manage the build process, while allowing you to define flexible custom build configurations.
- The **Gradle** will generate an apk from the **.java** and **.xml** files in the project. Simply put, a Gradle takes all the source files (java and XML) and apply appropriate tools, e.g., converts the java files into **dex** files and compresses all of them into a single file known as apk that is actually used.

## Hello World

- **Line 7**
- The **HelloWorld** activity is declared as a sub-class of **AppCompatActivity**
- **AppCompatActivity** is an indirect subclass of **Activity** class.
- **HelloWorld** class inherits all properties and methods from the **AppCompatActivity** and **Activity** class.
- **AppCompatActivity** class provides compatibility for different versions of the Android SDKs

**AppCompatActivity**

```

public class AppCompatActivity
    extends FragmentActivity implements AppCompatActivityCallback,
    TaskStackBuilder.SupportParentable, ActionBarDrawerToggle.DelegateProvider
    java.lang.Object
        ↳ android.content.Context
            ↳ android.content.ContextWrapper
                ↳ android.view.ContextThemeWrapper
                    ↳ android.app.Activity
                        ↳ androidx.activity.ComponentActivity
                            ↳ androidx.fragment.app.FragmentActivity
                                ↳ androidx.appcompat.app.AppCompatActivity

```

```

1 package com.sp.helloworld;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6
7 public class HelloWorld extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.main);
13    }
14}

```

# Hello World

- Line 10

When *HelloWorld* app is first launched, *HelloWorld* activity is created. At the beginning of the app lifecycle, ***onCreate*** callback method will be run and codes within this method will be executed first.

```

1  package com.sp.helloworld;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  public class HelloWorld extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.main);
13     }
14 }
```

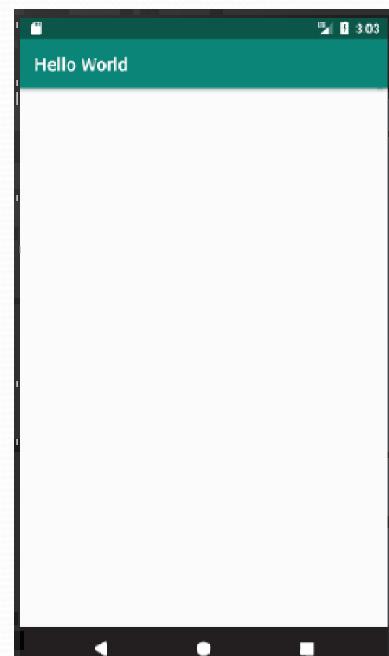
# Hello World

- Line 12

The ***setContentView(R.layout.main)*** will set the current display with the layout defined by *main.xml* in the *res/layout* folder

```

1  package com.sp.helloworld;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  public class HelloWorld extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.main);
13     }
14 }
```



# Hello World

- The *main.xml* layout defines the activity screen layout in XML.
- XML stands for **Extensible Markup Language**. It was designed to carry data

```

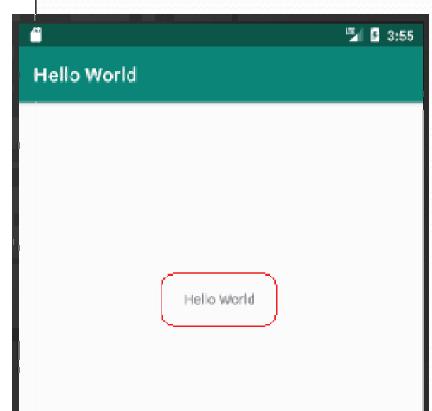
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HelloWorld">
9
10 </androidx.constraintlayout.widget.ConstraintLayout>
```

# Hello World

- Below is a re-designed *main.xml* layout that displays the “Hello World” message on the screen.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HelloWorld">
9
10 <TextView
11     android:id="@+id/textView"
12     android:layout_width="wrap_content"
13     android:layout_height="wrap_content"
14     android:text="Hello World"
15     app:layout_constraintBottom_toBottomOf="parent"
16     app:layout_constraintEnd_toEndOf="parent"
17     app:layout_constraintHorizontal_bias="0.498"
18     app:layout_constraintStart_toStartOf="parent"
19     app:layout_constraintTop_toTopOf="parent"
20     app:layout_constraintVertical_bias="0.327" />
21
22 </androidx.constraintlayout.widget.ConstraintLayout>
```



# Hello World

- XML documents must contain a **root element**. This element is "the parent" of all other elements.
- The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.
- Each element will begin with a "**start**" tag and will end with a "**end**" tag
- In between the "start" and "end" tag, you can have text content or more elements ("child" elements)
- In each element, **attributes** can be added to define the characteristic of the element

# Hello World

- Line 2 is start tag and line 22 is the end tag of the root element. It has a child element "TextView" with the start tag at line 10 and end tag at line 20.
- Line 3, 4, 5, 6, 7 and 8 are attributes of the root element.
- Line 11, 12, 13, 14, 15, 16, 17, 18, 19 and 20 are attributes of the "TextView" element.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HelloWorld">
9
10     <TextView
11         android:id="@+id/textView"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="Hello World"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintEnd_toEndOf="parent"
17         app:layout_constraintHorizontal_bias="0.498"
18         app:layout_constraintStart_toStartOf="parent"
19         app:layout_constraintTop_toTopOf="parent"
20         app:layout_constraintVertical_bias="0.327" />
21
22 </androidx.constraintlayout.widget.ConstraintLayout>

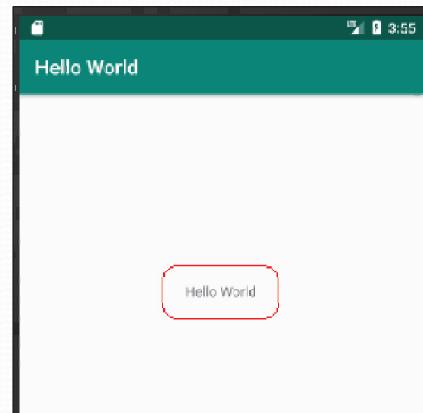
```

# Hello World

- When the new *main.xml* layout is referred, the layout will be inflated and the *TextView* widget will show a text “*Hello World*”

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HelloWorld">
9
10     <TextView
11         android:id="@+id/textView"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="Hello World"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintEnd_toEndOf="parent"
17         app:layout_constraintHorizontal_bias="0.498"
18         app:layout_constraintStart_toStartOf="parent"
19         app:layout_constraintTop_toTopOf="parent"
20         app:layout_constraintVertical_bias="0.327" />
21
22 </androidx.constraintlayout.widget.ConstraintLayout>
```



# Hello World

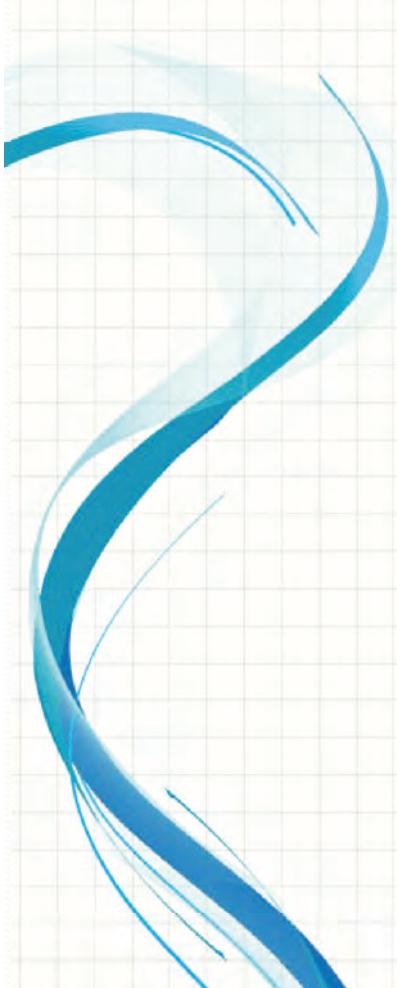
- In the *AndroidManifest.xml* file the *HelloWorld* activity is defined within the pair of tags *<activity>* and *</activity>*.
- HelloWorld* is also defined as the **main launch** Activity within the pair of tags *<intent-filter>* and *</intent-filter>*.
- A runtime exception will occur if trying to start any Activity not defined in the file

```

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Hello World"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.HelloWorld"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

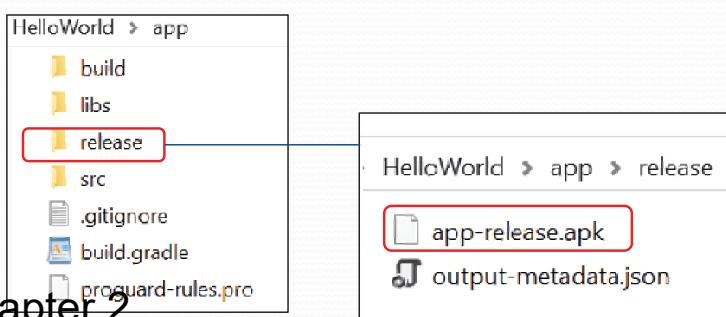


# Application Distribution File

Official (Closed), Non-Sensitive

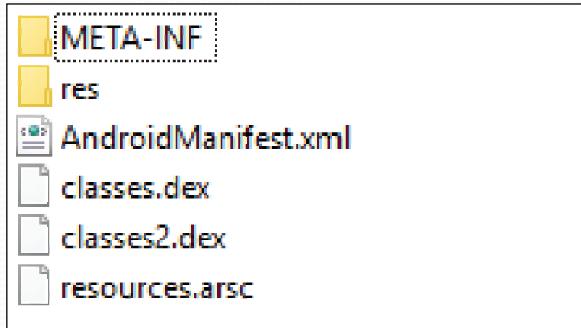
## Android Application Package (APK)

- Android **application package file (APK)** is the file format **used to distribute and install** application software **on to Android device**
- APK files are **ZIP file** formatted packages based on the JAR file format, with .apk the file extension.
- To create an APK for your project, watch  
<https://www.youtube.com/watch?v=3FujlwQoKuk>



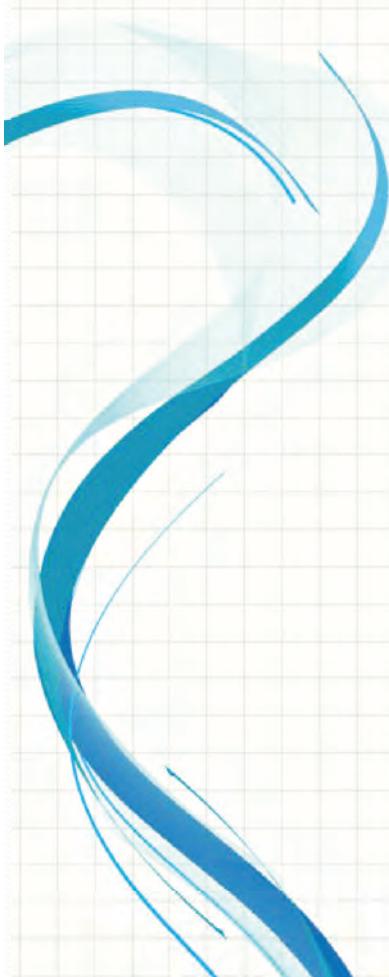
# Android Application Package (APK)

- An APK file is an archive that usually contains the following folders and files:
  1. **META-INF directory**
    - MANIFEST.MF: the Manifest file
    - CERT.RSA: The certificate of the application.
    - CERT.SF: The list of resources and SHA-1 digest
  2. **res directory** containing resources files e.g. images, icon.



# Android Application Package (APK)

3. **AndroidManifest.xml**: An additional Android manifest file, describing the name, version, access rights, referenced library files for the application
4. **classes.dex and classes2.dex**: The classes compiled in the dex format understandable by the ART and Dalvik virtual machine.
5. **resources.arsc**: a file containing pre-compiled resources in binary format; may include images, strings, or other data used by the program.



# Activity Lifecycle

Official (Closed), Non-Sensitive

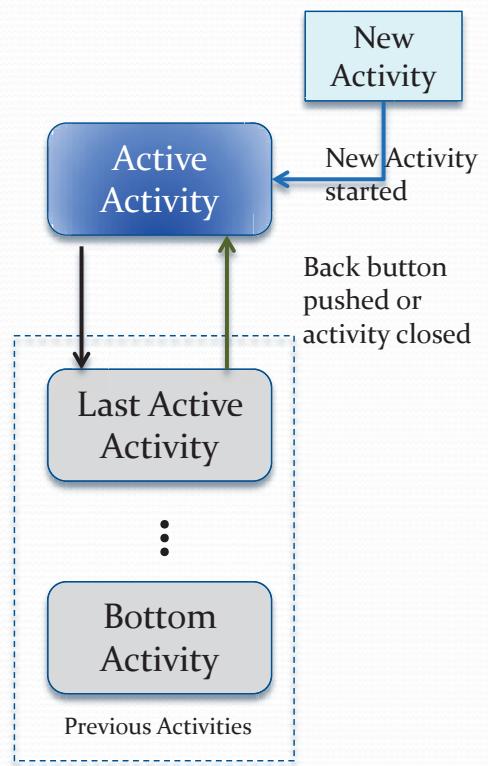
## Activity Lifecycle

- Android OS allows multiple apps to run concurrently
- There can be only one active app visible to user at a time – specifically, a single app Activity is in the foreground at any given time
- Android OS keeps track of all Activity objects running by placing them on an **Activity stack**

# Activity Lifecycle

- Activity Stack

- The state of each Activity is determined by its position on the Activity stack, a **last-in-first-out** collection **of all the currently running Activities**
- When a new Activity starts, the current foreground screen is moved to the top of the stack
- If the user navigates back using the BACK button, or the foreground Activity is closed, the next Activity on the stack moves up and becomes active.



# Activity Lifecycle

- As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle.
- The Activity class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.
- Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity.

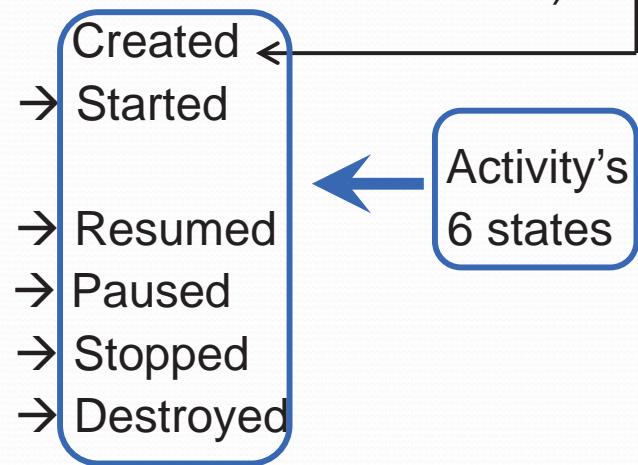
# Activity Lifecycle

- To ensure that Activities can react to state changes, Android provides a series of event handlers (or activity callbacks) that are fired when an Activity transitions through each stage of the life-cycle.

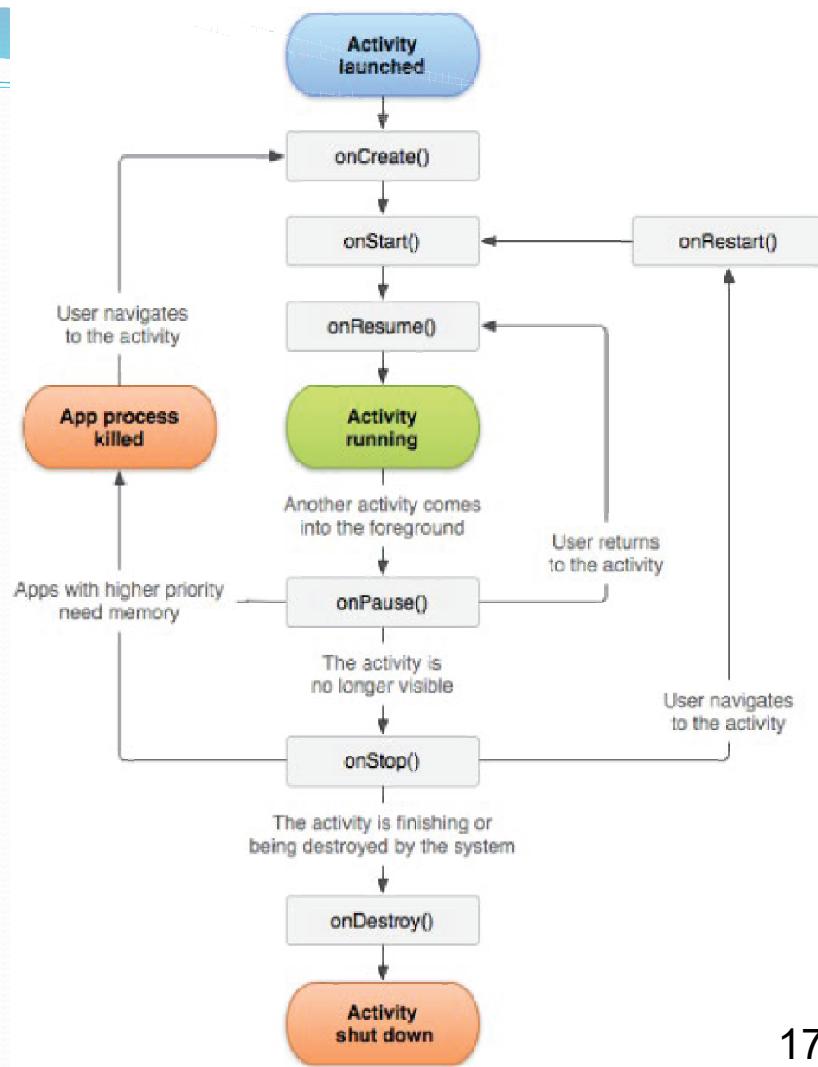
- Here are the **7 callback methods of Activity class:**

- protected void onCreate(Bundle savedInstanceState)

- protected void onStart()
- protected void onRestart()
- protected void onResume()
- protected void onPause()
- protected void onStop()
- protected void onDestroy()



# Activity Lifecycle



# Activity Lifecycle

- **onCreate(Bundle)**

- You must implement this callback, which fires when the system **first creates the activity**.
- In the `onCreate()` method, you perform basic application **startup logic** that should happen only **once for the entire life** of the activity.
- For example, your implementation of `onCreate()` might bind data to lists, associate the activity with a layout, and instantiate some class-scope variables.
- This method receives the parameter **`savedInstanceState`**, which is a **Bundle** object containing the activity's previously saved state. If the activity has never existed before, the value of the `Bundle` object is null.

```
protected void onCreate(Bundle savedInstanceState)
```

# Activity Lifecycle

- **onStart()**

- When the activity enters the Started state, the system invokes this callback. This method call when the activity **is becoming visible to the user** i.e. as the app **prepares** for the activity to enter the **foreground** and become interactive.
- It is also called when activity resumes from stopped state.
- For example, if you have activity A and starts activity B from it, then activity A will be paused (`onPause()`) and then stopped (`onStop()`) and moved to back stack. After this, if you press Back into your activity B, B will be paused(`onPause()`), stopped (`onStop()`) and destroyed(`onDestroy()`), and activity A will be restored from back stack, started (`onStart()`) and resumed(`onResume()`). As you can see, system will not call `onCreate()` for A again.

# Activity Lifecycle

- **onRestart()**
  - This method is called when a **stopped activity is brought to foreground again**. Android always calls onStart() after calling onRestart()
- **onResume()**
  - When the activity enters the Resumed state, **it comes to the foreground**, and then the system invokes the onResume() callback.
  - This is the **state in which the app interacts with the user**. The app stays in this state until something happens to take focus away from the app. Such an event might be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's turning off.

# Activity Lifecycle

- **onPause()**
  - The system calls this method as the first indication that the user is leaving your activity.
  - It indicates that the activity is no longer in the foreground e.g. when another activity takes focus (maybe as a pop up, or transparent window) while the current activity is still running or in a multi-window mode when the user change focus to another window.
  - However, if you navigate away from the app completely (for example, by hitting the home button), the activity is no longer visible then the system may execute onStop().

# Activity Lifecycle

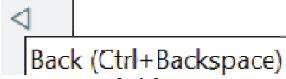
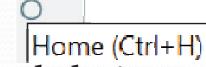
- `onStop()`
  - When your activity is **no longer visible to the user**, it has entered the **Stopped state**, and the system invokes the `onStop()` callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call `onStop()` when the activity has finished running, and is about to be terminated.
  - In the `onStop()` method, the app should **release or adjust resources** that are not needed while the app is not visible to the user. Using `onStop()` instead of `onPause()` ensures that UI-related work continues, even when the user is viewing your activity in multi-window mode.

# Activity Lifecycle

- `onDestroy()`
  - This method **is called before the activity is destroyed** because the activity is finishing, or because the **system is temporarily destroying this instance** of the activity to save space.

| Likelihood of being killed | Process state                             | Activity state                |
|----------------------------|-------------------------------------------|-------------------------------|
| Least                      | Foreground (having or about to get focus) | Created<br>Started<br>Resumed |
| More                       | Background (lost focus)                   | Paused                        |
| Most                       | Background (not visible)                  | Stopped                       |
|                            | Empty                                     | Destroyed                     |

# Activity Lifecycle—Try it out!

- Follow the guide – “Getting Started - Understanding Lifecycle” under Powerpoint Slides | Chapter 2
  - Add the rest 6 callback methods into your current Hello World project, e.g.: `protected void onPause(){...}`
  - Add toast in each of the 7 callback methods to print the method name, e.g.
    - `Toast.makeText(this, "onPause", Toast.LENGTH_LONG).show();`
  - Save, build and run the project
    - What's the toasted messages and their appearance sequence?
  - Click the back button 
    - What's the toasted messages and their appearance sequence?
  - Or, click the Home button 
    - What's the toasted messages and their appearance sequence?

# Application UI View Design



# Android Layout

Official (Closed), Non-Sensitive

## Android Layout

- Basically, user interface in Android apps is built using **layouts**.
- An Android layout is a type of resource that **defines what** is drawn **on the screen**. A layout resource is simply **a template for a user interface screen**, or portion of a screen
- Android user interfaces can be **defined as layout resources in XML or created programmatically**.
- Each layout definition is stored in as a **XML file**, in the **res/layout** folder. The filename then becomes the resource identifier.
  - R.layout.XMLFileName
  - **Please do not use the wrong XML layout file name for the activity**

# Android Layout

- The standard Layouts are:
  - ConstraintLayout
  - LinearLayout
  - TableLayout
  - FrameLayout
  - etc.

## ConstraintLayout

- ConstraintLayout is now the default layout in Android Studio.
- It is used to define a layout by assigning constraints for every child view/widget relative to other views present.
- It allows you to create large and complex layouts.
- A view inside the ConstraintLayout has handles(or anchor points) on each side which are used to assign the constraints.

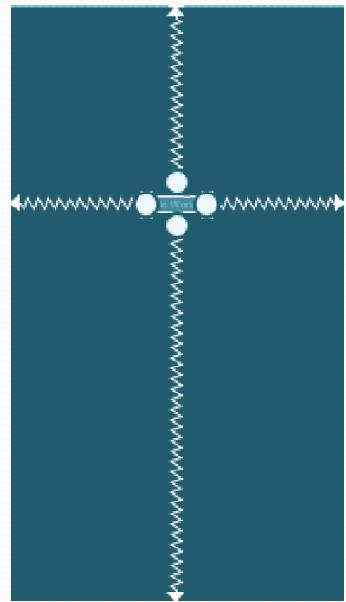
# ConstraintLayout

- Example

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HelloWorld">
9
10 <TextView
11     android:id="@+id/textView"
12     android:layout_width="wrap_content"
13     android:layout_height="wrap_content"
14     android:text="Hello World"
15     app:layout_constraintBottom_toBottomOf="parent"
16     app:layout_constraintEnd_toEndOf="parent"
17     app:layout_constraintHorizontal_bias="0.498"
18     app:layout_constraintStart_toStartOf="parent"
19     app:layout_constraintTop_toTopOf="parent"
20     app:layout_constraintVertical_bias="0.327" />
21
22 </androidx.constraintlayout.widget.ConstraintLayout>

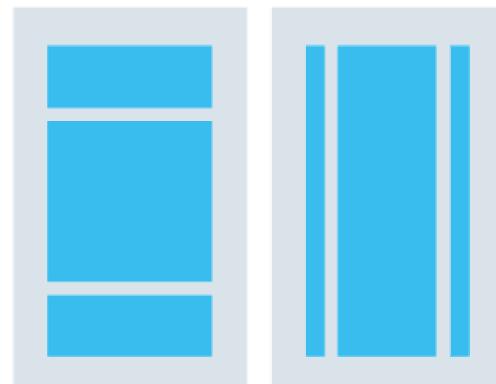
```



# LinearLayout

- LinearLayout is the most basic type of Layout.
- This type of Layout enforces you to put your controls in a linear direction, either **horizontally** or **vertically**.
- You can specify the layout direction with the **android:orientation** attribute.

**Vertical  
LinearLayout**      **Horizontal  
LinearLayout**



# LinearLayout

- Example

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

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

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

    <Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 3" />
</LinearLayout>
```

Linear Layout (Vertical)



# LinearLayout

- Example

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

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

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

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3" />
</LinearLayout>
```

Linear Layout (Horizontal)



# TableLayout

- TableLayout arranges its children into rows and columns.
- It consists of a number of TableRow objects, each defining a row.
- TableLayout containers do not display border lines for their rows, columns, or cells.
- Each row has zero or more cells; each cell can hold one View object.
- The table has as many columns as the row with the most cells.
- A table can leave cells empty. Cells can span columns, as they can in HTML.

# TableLayout

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

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

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 2" />
    </TableRow>

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 3" />

        <Button
            android:id="@+id/button4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 4" />
    </TableRow>
</TableLayout>
```

Table Layout

|          |          |
|----------|----------|
| BUTTON 1 | BUTTON 2 |
| BUTTON 3 | BUTTON 4 |

# FrameLayout

- FrameLayout is designed **to display a single view at a time.**
- You can have multiple **views** within a FrameLayout but **each view will be positioned based on the top left of the screen.**
- All the child views added are placed like **stack**. The most recent added are shown on top. Hence the order of elements in the layout is of importance.
- It is used to display only one view, or views which overlap.

# FrameLayout

- FrameLayout can become more **useful when elements are hidden and displayed programmatically.**
- You can use the attribute **android:visibility** in the XML to hide specific elements.
- You can call **setVisibility** from the code to accomplish the same thing. The three available visibility values are **visible**, **invisible** (does not display, but still takes up space in the layout), and **gone** (does not display, and does not take space in the layout)

# FrameLayout

- Example

The '**Button**' view is the most recent added to the layout, it shown overlapped the '**Hello World**' view

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello World"
        android:textAlignment="viewStart"
        android:textSize="36sp" />

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



# UI and Layout Design

It's Quiz time

- The top most layout is
  - VerticalLinearLayout
  - FrameLayout
  - ConstraintLayout

# CREATING USER INTERFACES

Official (Closed), Non-Sensitive

## Today's Overview

1

- Model-View-Controller

2

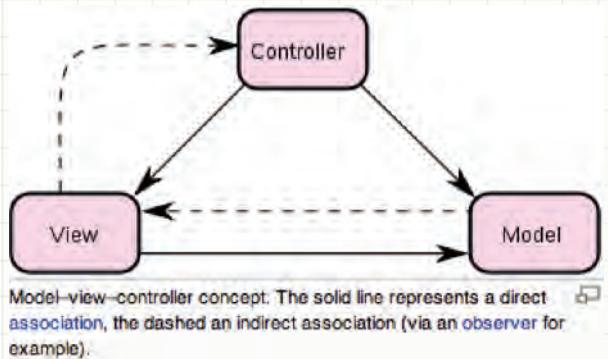
- Practical 1 Walk Through

# Model View Controller (MVC)

Official (Closed), Non-Sensitive

## Model-View-Controller (MVC)

- MVC is a **software design pattern** that separates an application into three main logical components:
  - *Model*
  - *View*
  - *Controller*
- Each of these components are built to handle specific development aspects of an application.



# Model-View-Controller (MVC)

## Model

- Corresponds to all the **data-related logic** that the user works with.
- It will **contain functions** that **help** you **read, insert, delete and update information**.
- It is **responsible for managing the data of the application**.
- For example, a Customer object will retrieve the customer information from the database (**model**), manipulate it (**controller**) and update its data back to the database (**model**) or use it to render data (**view**).

# Model-View-Controller (MVC)

## View

- It is used for all the **UI logic** of the application.
- It has UI components that **present information to the user or gather inputs** from the user.
- For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

# Model-View-Controller (MVC)

## View

- Through the **Controller**, **View** pulls data from the **Model** and present the data to the user.
- The view does not cause the data to be modified in any way, it only displays data retrieved from the model.

# Model-View-Controller (MVC)

## Controller

- It act as an interface between **Model** and **View** components **to process all the business logic and incoming requests**, manipulate the data (using **read, insert, delete and update functions**) using the Model component and interact with the Views to render the final output.
- It handles and responds to user input and interaction.

# Model-View-Controller (MVC)

## Controller

- It is **responsible for responding to user actions (incoming requests).**
- The controller will **determine** what **request** is being made **by the user** and **respond** appropriately **by triggering the model to manipulate the data** appropriately and **passing the model to the view for presentation.**

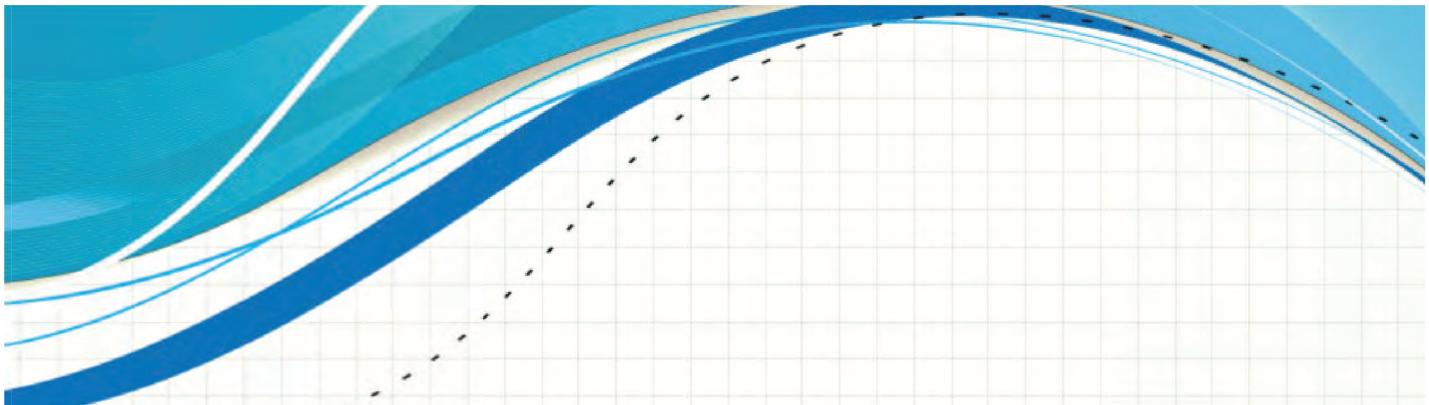
## It's Quiz time!

1. In MVC, Model refers to:

- A. Data
- B. UI
- C. Presentation
- D. Design

2. In MVC, the middle man(the glue) is:

- A. Model
- B. View
- C. Controller
- D. None of these



# RESTAURANT LIST

Official (Closed), Non-Sensitive

## Restaurant List

- Restaurant List is an Android app to allow user to store restaurant information to a local database. It provides features:
  - to input restaurant information and save to database
  - to display the restaurant information from the database
  - to access location coordinates and save to database
  - to show restaurant location on Google Map with a marker

# Restaurant List

- The 5 practical sessions will provide you a step-by-step guide in building the application
- The approach will be following the Model-View-Controller software design pattern

## Practical 1 – UI View

# User Interface View Design

- Applications are usually made up of Views. They provide interaction between user and the app.
- The basic function of these views is to take inputs from users or present outputs to them.

## User Interface View Design

- For the beginning, a Simple Form will be designed for the UI View using
    - LinearLayout
    - TextView
    - EditText
    - Button
- 

```

java.lang.Object
↳ android.view.View
    ↳ android.view.ViewGroup
        ↳ android.widget.LinearLayout
  
```

```

java.lang.Object
↳ android.view.View
    ↳ android.widget.TextView
        ↳ android.widget.EditText
  
```

```

java.lang.Object
↳ android.view.View
    ↳ android.widget.TextView
        ↳ android.widget.Button
  
```

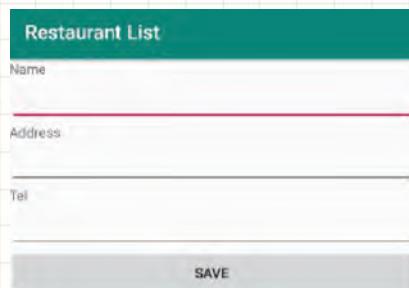
# User Interface View Design

- Simple Form

- Set the layout to hold the view design.

- Place in the necessary widgets:

- TextView – to display label to user
- EditText – to read data from user
- Button – to get user's request



# User Interface View Design

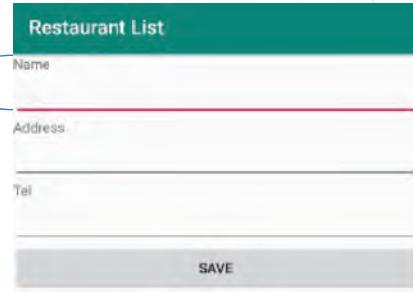
```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RestaurantList">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Name" />

        <EditText
            android:id="@+id/restaurant_name"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPersonName"
            android:maxLength="30" />
    
```



# User Interface View Design

```

<TextView
    android:id="@+id/textView2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Address" />

<EditText
    android:id="@+id/restaurant_address"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPostalAddress"
    android:maxLength="60" />

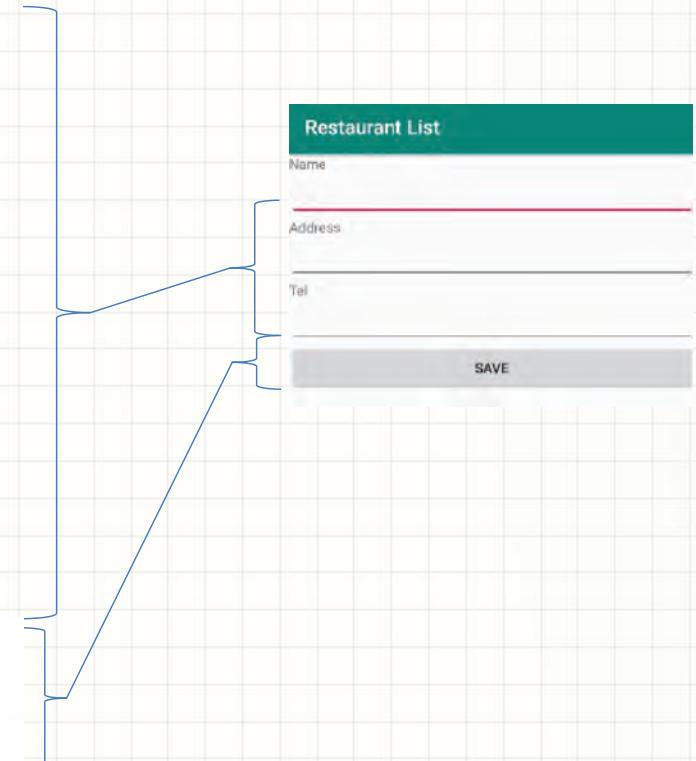
<TextView
    android:id="@+id/textView3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Tel" />

<EditText
    android:id="@+id/restaurant_tel"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="phone"
    android:maxLength="8" />

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

</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>

```



## UI View Design

### ConstraintLayout

- **xmlns:android** attribute defines the XML name space to be refer to.
- XML Namespaces provide a method to avoid element name conflicts.

```

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RestaurantList">

```

- **android:layout\_width** and **android:layout\_height** has the attribute values “**match\_parent**” set the layout to as big as its parent.
- More about ConstraintLayout - <https://developer.android.com/training/constraint-layout>

# UI View Design

## LinearLayout

- android:layout\_width and android:layout\_height has the attribute values “match\_parent” set the layout to as big as its parent; in this case the “ConstraintLayout”.
- android:orientation=“vertical” sets the LinearLayout to single column.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Name" />

```

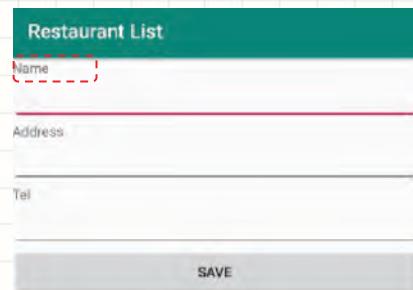
- More about LinearLayout - <https://developer.android.com/guide/topics/ui/layout/linear>

# UI View Design

## TextView

- A user interface element to display text to the user.
- Setting android:layout\_height = “wrap\_content” specify that the basic height of the TextView should be only big enough to enclose its content (plus padding)
- android:text sets the text displayed.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Name" />
```



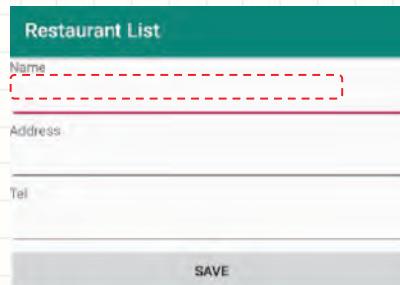
- More about TextView - <https://developer.android.com/reference/android/widget/TextView>

# UI View Design

## EditText

- A user interface element for entering and modifying text.
- `android:id = "@+id/restaurant_name"` creates an **id** named **restaurant\_name** for the widget. Within the app, **id** must be **unique**.
- This will allow the **app to access** to it through this **id**.

```
<EditText
    android:id="@+id/restaurant_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:maxLength="30" />
```

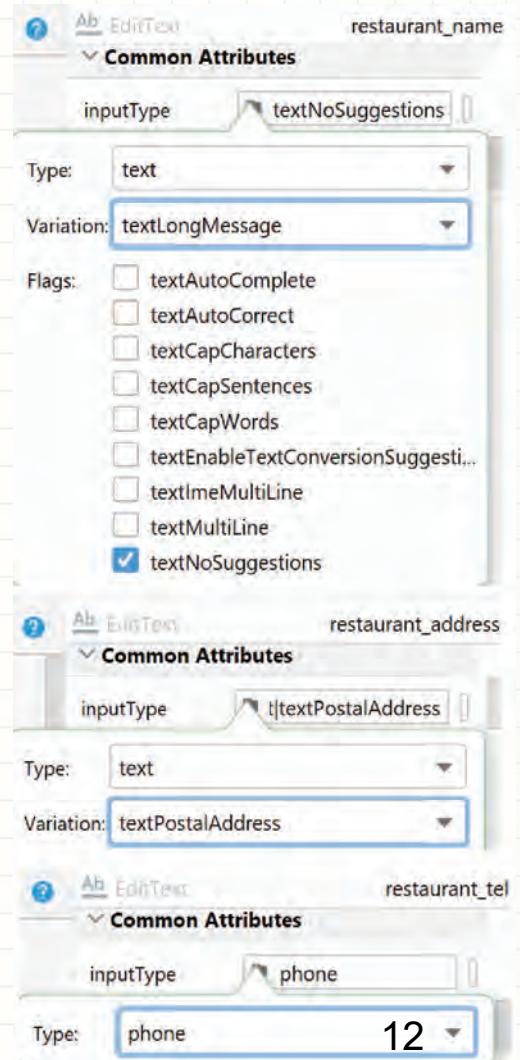


- More about **EditText** -  
<https://developer.android.com/reference/android/widget/EditText>

# UI View Design

## EditText

- `android:ems="10"` sets to 10 em units (optional)  
 (Refer to  
<https://www.quora.com/What-is-EMS-in-Android-XML> for more detail about em)
- You should always declare the **input method** for your text fields by adding the `android:inputType` attribute to the **EditText** element.
- Choosing the input type configures the keyboard type that is shown, acceptable characters, and appearance of the edit text.



# UI View Design

## EditText

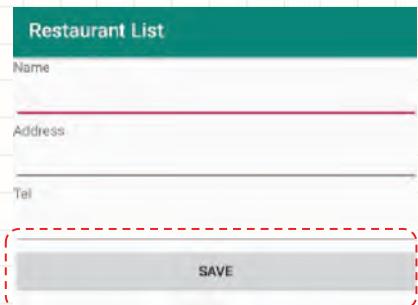
- For example, if you want to accept a secret number, like a unique pin or serial number, you can set `inputType` to "numericPassword". An `inputType` of "numericPassword" results in an edit text that accepts numbers only, shows a numeric keyboard when focused, and masks the text that is entered for privacy.
- `android:inputType="textPersonName"` set the input type to text input to enter name with no spelling check.
- More about `inputType` -  
<https://developer.android.com/training/keyboard-input/style>

# UI View Design

## Button

- A user interface element the user can tap or click to perform an `action`.
- `android:id = "@+id/button_save"` creates an id named `button_save` in the namespace of the app.
- More about Button -  
<https://developer.android.com/reference/android/widget/Button>

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

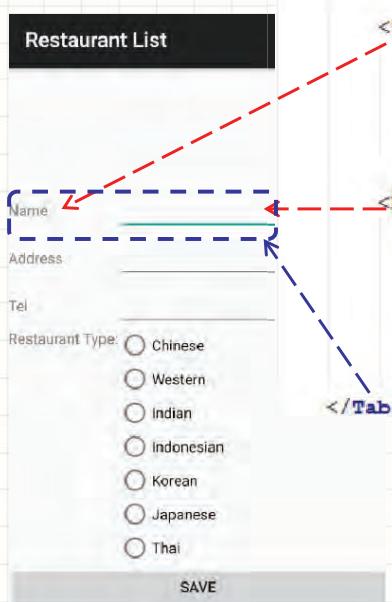


# Practical 1 – Fancier UI View

Official (Closed), Non-Sensitive

## User Interface View Design

- Instead of using LinearLayout with single column, **TableLayout** is used to improve the view design by arranging widgets in columns for optimization.

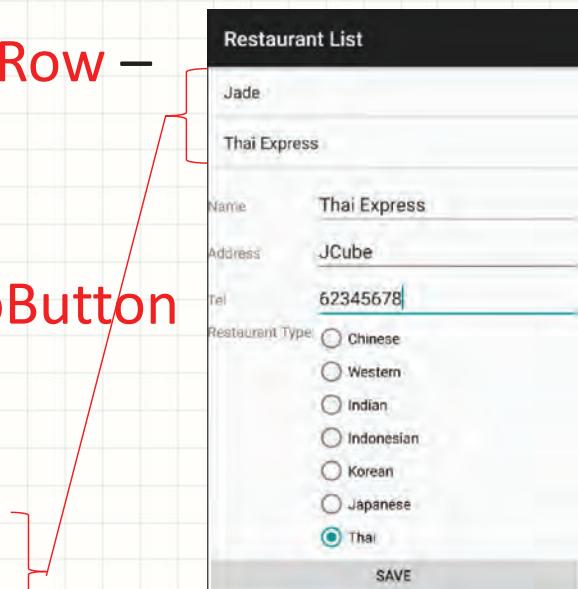


```
<TableLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:stretchColumns="1">  
  
<TableRow  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
  
    <TextView  
        android:id="@+id/textView"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Name" />  
  
    <EditText  
        android:id="@+id/restaurant_name"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:ems="10"  
        android:inputType="textPersonName"  
        android:maxLength="30" />  
  
</TableRow>
```

# User Interface View Design

## The new form includes

- **TableLayout** and **TableRow** – to arrange widgets in columns
- **RadioGroup** and **RadioButton** – for restaurant type selection
- **ListView** – to keep the restaurant records



# User Interface View Design

## TableLayout

- A layout that arranges its children into **rows** and **columns** and it consists of a number of **TableRow** objects.
- A **TableRow** object define row.
- TableLayout containers do not display border lines for their rows, columns, or cells.

```

<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Name" />

        <EditText
            android:id="@+id/restaurant_name"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPersonName"
            android:maxLength="30" />
    </TableRow>

```

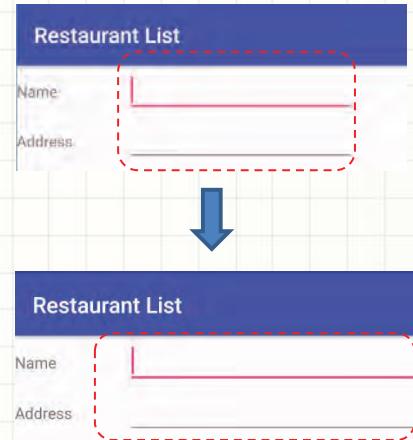
# User Interface View Design

## TableLayout

- **android:stretchColumn="1"**  
stretches the EditText widget which is at column 1 to occupy the full column width.

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1">
```

- More about TableLayout - <https://developer.android.com/reference/android/widget/TableLayout>



# User Interface View Design

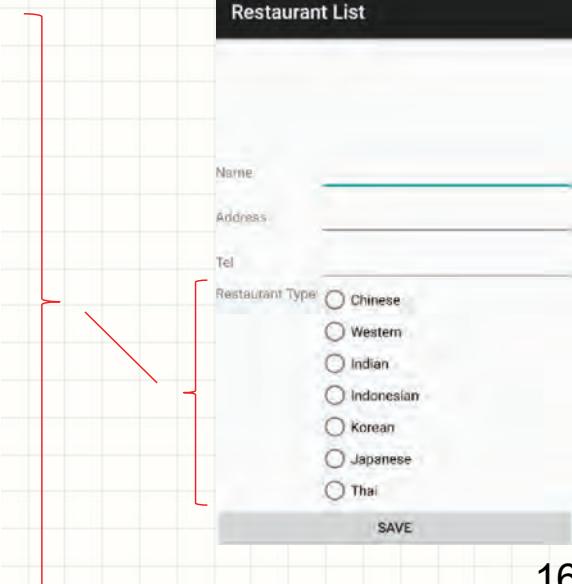
## RadioGroup

- It is used to create a **multiple-exclusion scope** for a set of radio buttons.
- Checking one radio button that belongs to a radio group unchecks any previously checked radio button within the same group.

```
<RadioGroup
    android:id="@+id/restaurant_types"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <RadioButton
        android:id="@+id/chinese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Chinese" />

    <RadioButton
        android:id="@+id/western"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Western" />
```



# User Interface View Design

## RadioButton

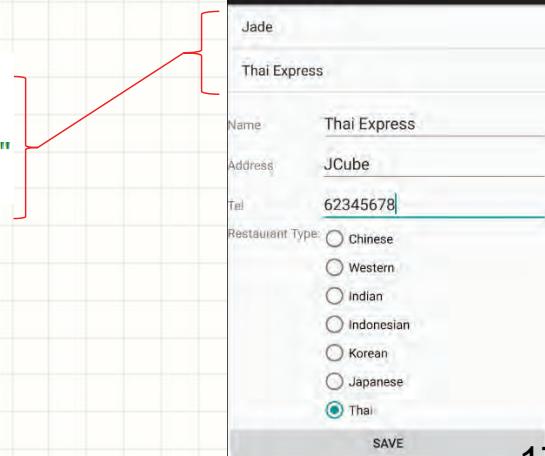
- More about RadioGroup - <https://developer.android.com/reference/android/widget/RadioGroup>
- A radio button is a **two-states** button that can be **either checked or unchecked**. When the radio button is unchecked, the user can press or click it to check it.
- Radio buttons are normally used together in a **RadioGroup**. When several radio buttons live inside a radio group, checking one radio button unchecks all the others.
- More about RadioButton - <https://developer.android.com/reference/android/widget/RadioButton>

# User Interface View Design

## ListView

- A view that shows items in a vertically scrolling list. The items displayed come from a **ListAdapter** associated with this list view.
- More about ListView - <https://developer.android.com/reference/android/widget/ListView>

```
<ListView
    android:id="@+id/restaurants"
    android:layout_width="match_parent"
    android:layout_height="110dp" />
```



# It's Quiz time!

1. In MVC, Model refers to:
  - A. Data
  - B. UI
  - C. Presentation
  - D. Design
  
2. In MVC, the middle man(the glue) is:
  - A. Model
  - B. View
  - C. Controller
  - D. None of these



# Practical 1 – Model

# Model

- With the view design ready, we want to add a **Model**. The class *Restaurant.java* is the **Model** to hold a single restaurant data
- This **Model** contains a set of variables to store **data** and **functions** to read (**getter**) and set (**setter**) the data.

```
package com.sp.restaurantlist;

public class Restaurant {
    private String restaurantName = "";
    private String restaurantAddress = "";
    private String restaurantTel = "";
    private String restaurantType = "";

    public String getName() { return restaurantName; }

    public void setName(String restaurantName) { this.restaurantName = restaurantName; }

    public String getAddress() { return restaurantAddress; }

    public void setAddress(String restaurantAddress) { this.restaurantAddress = restaurantAddress; }

    public String getTelephone() { return restaurantTel; }

    public void setTelephone(String restaurantTel) { this.restaurantTel = restaurantTel; }

    public String getRestaurantType() { return restaurantType; }

    public void setRestaurantType(String restaurantType) { this.restaurantType = restaurantType; }

    @Override
    public String toString() { return (getName()); }
}
```

# Model

## *restaurant.java* Class

- Four local data variables of **String** data type are created to store the **name**, **address**, **telephone** and the **restaurant type** of a restaurant in the restaurant model. Each variable is initialised to a blank string.

```
private String restaurantName = "";
private String restaurantAddress = "";
private String restaurantTel = "";
private String restaurantType = "";
```

Name: \_\_\_\_\_

Address: \_\_\_\_\_

Tel: \_\_\_\_\_

Restaurant Type:

- Chinese
- Western
- Indian
- Indonesian
- Korean
- Japanese
- Thai

SAVE

# Model

- Methods for writing data into local variables (setter functions)

```

public void setName(String restaurantName) {
    this.restaurantName = restaurantName;
}

public void setAddress(String restaurantAddress) {
    this.restaurantAddress = restaurantAddress;
}

public void setTelephone(String restaurantTel) {
    this.restaurantTel = restaurantTel;
}

public void setRestaurantType(String restaurantType) {
    this.restaurantType = restaurantType;
}

```

is assigned to private data via setter method

Parameter value from outside

# Model

- Methods for reading data from local variables (getter functions)

```

public String getName() { return (restaurantName); }
public String getAddress() { return (restaurantAddress); }
public String getTelephone() { return (restaurantTel); }
public String getRestaurantType() { return (restaurantType); }

```

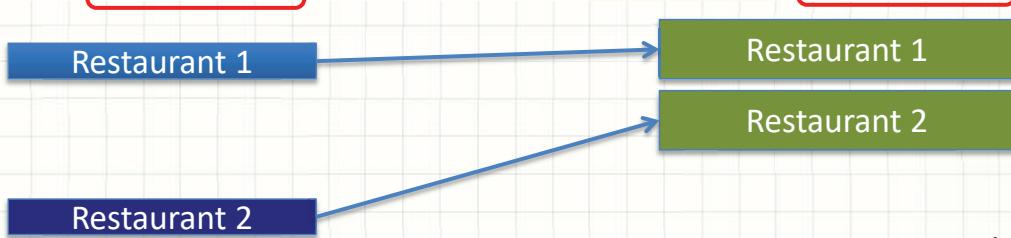
# Model

- The Restaurant model created is only for keeping a single restaurant data. What will happen when we have a list of restaurants?
- We will need a bigger model to keep a list of restaurants.
- We will use an *ArrayList* model.

# Model

- Data of individual restaurant will now be able to be kept in a *ArrayList* model in listing format

```
List<Restaurant> model = new ArrayList<Restaurant>();
```



ArrayList Restaurant Model

Individual Restaurant Model

# It's Quiz Time!

1. In Practical 1, the model is:

- A. Restaurant.java class only
- B. ArrayList class only
- C. Both Restaurant and ArrayList
- D. Neither Restaurant.java nor ArrayList

2. The relationship between Restaurant class and ArrayList class used in Practical 1 is:

- A. ArrayList object is used to hold multiple Restaurant object.
- B. Restaurant object is used to hold multiple ArrayList object.

3. In Restaurant.java class, the restaurant data is stored in:

- A. 4 private String type of variables.
- B. 4 private EditText type of variables.
- C. 4 getter methods
- D. 4 setter methods.

# It's Quiz Time!

4. In Restaurant.java class of Practical 1 :

- A. The getter methods is to retrieve the private variable's value.
- B. The setter methods is to retrieve the private variable's value.
- C. The getter methods is to store external value to the Restaurant object's private variable.
- D. The setter methods is to store external value to the Restaurant object's private variable.

5. To create a Restaurant object:

- A. Restaurant r = new Restaurant;
- B. Restaurant r = Restaurant();
- C. Restaurant r = new Restaurant();
- D. Restaurant r = new Restaurant("Sakura");

# It's Quiz Time!

6. Refer to the statement below:

- A. The object name is model.
- B. The class name is List.
- C. The constructor name is ArrayList.
- D. The ArrayList's element is of Restaurant data type.

# Controller

# Controller

- We have
  - a **UI View** for user interaction and
  - a **Model** for store a restaurant record.
- To interface the **Model** and **View** components we need the **Controller**.
- The **Controller** will act as a “**middle man**” in between the **UI View (for user)** and **Model (for program)**
- It **monitors requests from View** e.g. Save button is clicked and **takes the necessary action with the model provided**



# Controller

- Referring to the **RestaurantList.java**, when ***RestaurantList Activity*** first started, the **onCreate(Bundle)** callback method will be run.
- The **onCreate** method is only run once to initialise the activity.
- Here you will usually call **setContentView(int)** with a layout resource defining your UI, and using **findViewById(int)** to retrieve the widgets in that UI that you need to interact with programmatically.
- Here you might perform **basic application startup logic** that should happen only once for the entire life of the activity e.g. bind data to lists, instantiate some class-scope variables.

# Controller

```

16 public class RestaurantList extends AppCompatActivity {
17     private EditText restaurantName;
18     private RadioGroup restaurantTypes;
19     private Button buttonSave;
20     private EditText restaurantAddress;
21     private EditText restaurantTel;
22
23     private List<Restaurant> model = new ArrayList<Restaurant>();
24     private ArrayAdapter<Restaurant> adapter = null;
25     private ListView list;
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.main);
31
32         restaurantName = findViewById(R.id.restaurant_name);
33         restaurantTypes = findViewById(R.id.restaurant_types);
34
35         buttonSave = findViewById(R.id.button_save);
36         buttonSave.setOnClickListener(onSave);
37
38         restaurantAddress = findViewById(R.id.restaurant_address);
39         restaurantTel = findViewById(R.id.restaurant_tel);
40
41         list = findViewById(R.id.restaurants);
42         adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, model);
43         list.setAdapter(adapter);
44     }

```

# Controller

- Line 17 to 25 – declare the widget local variables that will binds to EditText, RadioGroup, Button and ListView widgets on UI View defined in *main.xml*

```

16 public class RestaurantList extends AppCompatActivity {
17     private EditText restaurantName;
18     private RadioGroup restaurantTypes;
19     private Button buttonSave;
20     private EditText restaurantAddress;
21     private EditText restaurantTel;
22
23     private List<Restaurant> model = new ArrayList<Restaurant>();
24     private ArrayAdapter<Restaurant> adapter = null;
25     private ListView list;
26

```

# Controller

- Line 30 – “setContentView” sets display view layout using *main.xml*
- Line 32 to 41 – “findViewById” binds instance variables to EditText, RadioGroup, Button and ListView widgets on UI View defined in *main.xml*

```

27
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.main);
31
32         restaurantName = findViewById(R.id.restaurant_name);
33         restaurantTypes = findViewById(R.id.restaurant_types);
34
35         buttonSave = findViewById(R.id.button_save);
36         buttonSave.setOnClickListener(onSave);
37
38         restaurantAddress = findViewById(R.id.restaurant_address);
39         restaurantTel = findViewById(R.id.restaurant_tel);
40
41         list = findViewById(R.id.restaurants);
42         adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,model);
43         list.setAdapter(adapter);
44     }

```

# Controller

- Line 35 - bind the Button variable “buttonSave” to the “Save” button widget on UI View with id “button\_save”.
- Line 36 - activate the click listener to monitor the touch event of the button.

```

27
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.main);
31
32         restaurantName = findViewById(R.id.restaurant_name);
33         restaurantTypes = findViewById(R.id.restaurant_types);
34
35         buttonSave = findViewById(R.id.button_save);
36         buttonSave.setOnClickListener(onSave);
37
38         restaurantAddress = findViewById(R.id.restaurant_address);
39         restaurantTel = findViewById(R.id.restaurant_tel);
40
41         list = findViewById(R.id.restaurants);
42         adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,model);
43         list.setAdapter(adapter);

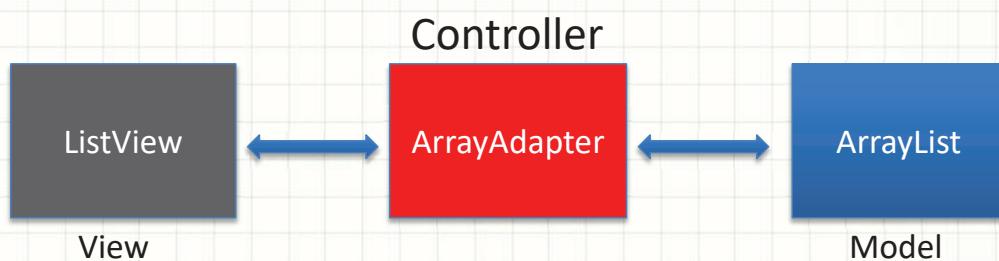
```

# ArrayAdapter

Official (Closed), Non-Sensitive

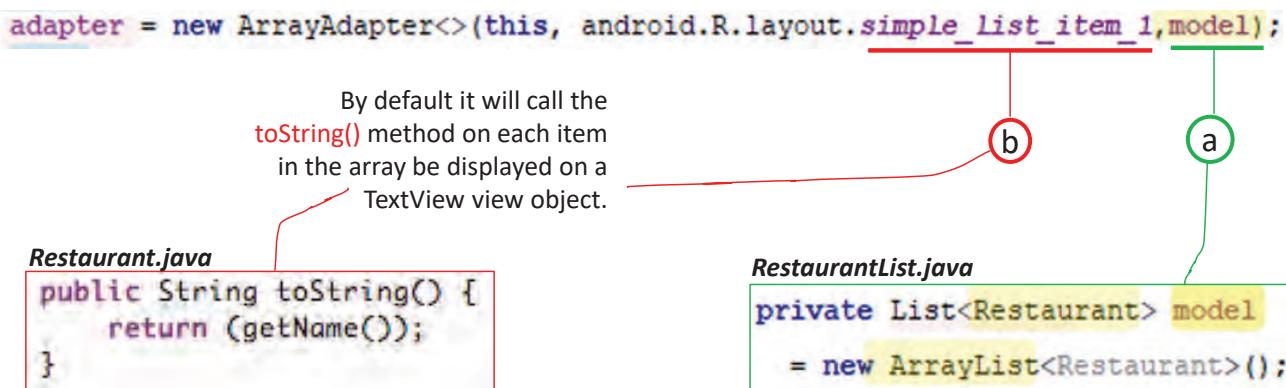
## ArrayAdapter

- Remember that the **Restaurant model** only store a single **record** of a restaurant.
- To store multiple records we will use an **ArrayList Model**. This model will store multiple restaurant records as a **list**.
- To view all the restaurant records, we will use the **ListView UI widget** that show a vertical list of scrollable items.
- To link the **ArrayList model** with the **ListView widget** on the **UI View**, the Controller **uses ArrayAdapter** to bind the two together.



# ArrayAdapter

- The **ArrayAdapter** fits in between an **ArrayList** (data source) and the **ListView**  
`list.setAdapter(adapter);`
- It configures two aspects:
  - Which array to use as the data source for the list?
  - How to convert any given item in the array into a corresponding View object?



Official (Closed), Non-Sensitive

# ArrayAdapter

- Line 42 - an **ArrayAdapter** is declared to link to **ArrayList** **model** with Android predefined layout **android.R.layout.simple\_list\_item\_1**

```
27     @Override  
28     protected void onCreate(Bundle savedInstanceState) {  
29         super.onCreate(savedInstanceState);  
30         setContentView(R.layout.main);  
31  
32         restaurantName = findViewById(R.id.restaurant_name);  
33         restaurantTypes = findViewById(R.id.restaurant_types);  
34  
35         buttonSave = findViewById(R.id.button_save);  
36         buttonSave.setOnClickListener(onSave);  
37  
38         restaurantAddress = findViewById(R.id.restaurant_address);  
39         restaurantTel = findViewById(R.id.restaurant_tel);  
40  
41         list = findViewById(R.id.restaurants);  
42         adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, model);  
43         list.setAdapter(adapter);  
44     }
```

# ArrayAdapter

- Line 43 - bind the *ArrayList* model and *ListView* widget with *ArrayAdapter*

```

27
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.main);
31
32         restaurantName = findViewById(R.id.restaurant_name);
33         restaurantTypes = findViewById(R.id.restaurant_types);
34
35         buttonSave = findViewById(R.id.button_save);
36         buttonSave.setOnClickListener(onSave);
37
38         restaurantAddress = findViewById(R.id.restaurant_address);
39         restaurantTel = findViewById(R.id.restaurant_tel);
40
41         list = findViewById(R.id.restaurants);
42         adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,model);
43         list.setAdapter(adapter);
44     }

```

# Controller

- Line 36 set the button, *buttonSave* is set to listen for the button *onClick* event of the “*onSave*” *OnClickListener*.
- When Save Button is pressed, the Controller will handle the event by executing the code in *onClick* method defined in the *OnClickListener* “*onSave*” method.

```

35             buttonSave = findViewById(R.id.button_save);
36             buttonSave.setOnClickListener(onSave);
37
38
39
40
41
42
43
44
45
46     private View.OnClickListener onSave = new View.OnClickListener() {
47         @Override
48         public void onClick(View v) {
49             // To read data from restaurantName EditText
50             String nameStr = restaurantName.getText().toString();
51             // To read data from restaurantAddress EditText
52             String addressStr = restaurantAddress.getText().toString();
53             // To read data from restaurantTel EditText
54             String telStr = restaurantTel.getText().toString();
55             String restType = "";
56             //To read selection of restaurantTypes RadioGroup

```



# Controller

- Line 49 to 54 - Get the data entries – restaurant name, address and telephone from the UI EditText Views and save to local variables nameStr, addressStr and telStr respectively.

```
46     private View.OnClickListener onSave = new View.OnClickListener() {
47
48     @Override
49     public void onClick(View v) {
50         // To read data from restaurantName EditText
51         String nameStr = restaurantName.getText().toString();
52         // To read data from restaurantAddress EditText
53         String addressStr = restaurantAddress.getText().toString();
54         // To read data from restaurantTel EditText
55         String telStr = restaurantTel.getText().toString();
56         String restType = "";
57         //To read selection of restaurantTypes RadioGroup
58         int radioID = restaurantTypes.getCheckedRadioButtonId();
59         if (radioID == R.id.chinese ) {
60             restType = "Chinese";
61         } else
```

# Controller

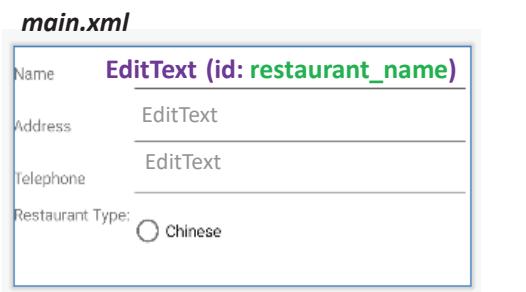
- For instance, Line 50 - Get the *restaurant name* entered by the user and store it in *nameStr* variable.

```
46     private View.OnClickListener onSave = new View.OnClickListener() {
47
48     @Override
49     public void onClick(View v) {
50         // To read data from restaurantName EditText
51         String nameStr = restaurantName.getText().toString();
52         // To read data from restaurantAddress EditText
53         String addressStr = restaurantAddress.getText().toString();
54         // To read data from restaurantTel EditText
55         String telStr = restaurantTel.getText().toString();
56         String restType = "";
57         //To read selection of restaurantTypes RadioGroup
58         int radioID = restaurantTypes.getCheckedRadioButtonId();
59         if (radioID == R.id.chinese ) {
60             restType = "Chinese";
61         } else
```

# Controller

- Line 50 is possible because

Firstly, an EditText is being defined in *main.xml* with id of `restuarant_name`



*RestaurantList.java/onCreate()*

```
private EditText restaurantName;  
restaurantName = findViewById(R.id.restaurant_name);
```

Next, in *RestaurantList.java* the `restuarant_name` is linked (bind) to a variable `restaurantName`. From here onwards, this variable `restaurantName` will be used in other parts of the program.

*RestaurantList.java/onSave() – Line 50*

```
String nameStr = restaurantName.getText().toString();
```

`restaurantName.getText().toString()` reads in data entered by the user

Official (Closed), Non-Sensitive

# Controller

- Line 57 to 78 - Get the data entry – restaurant type from the UI RadioGroup View and save to local variable `restType`.

```
56 //To read selection of restaurantTypes RadioGroup  
57 int radioID = restaurantTypes.getCheckedRadioButtonId();  
58 if (radioID == R.id.chinese ) {  
59     restType = "Chinese";  
60 } else  
61     if (radioID == R.id.western ) {  
62         restType = "Western";  
63 } else  
64     if (radioID == R.id.indian ) {  
65         restType = "Indian";  
66 } else  
67     if (radioID == R.id.indonesian ) {  
68         restType = "Indonesian";  
69 } else  
70     if (radioID == R.id.korean) {  
71         restType = "Korean";  
72 } else  
73     if (radioID == R.id.japanese) {  
74         restType = "Japanese";  
75 } else  
76     if (radioID == R.id.thai) {  
77         restType = "Thai";  
78 }
```

# Controller

- Line 83 - Create a *restaurant* data model. It will store a record of a restaurant.
- Line 85 to 88 – Store the restaurant name, address, telephone and the restaurant type in the data model using the setter methods defined in the Restaurant model.

```

82      //Create Restaurant Data Model
83      Restaurant restaurant = new Restaurant();
84      //Update the Restaurant Data Model
85      restaurant.setName(nameStr);
86      restaurant.setAddress(addressStr);
87      restaurant.setTelephone(telStr);
88      restaurant.setRestaurantType(restType);
89      //Pass the record to ArrayAdapter.
90      //It will update the ListArray and the ListView
91      adapter.add(restaurant);

```

# Controller

- Line 91 - Add the new *restaurant* data model record to the *ArrayAdapter*.

```

82      //Create Restaurant Data Model
83      Restaurant restaurant = new Restaurant();
84      //Update the Restaurant Data Model
85      restaurant.setName(nameStr);
86      restaurant.setAddress(addressStr);
87      restaurant.setTelephone(telStr);
88      restaurant.setRestaurantType(restType);
89      //Pass the record to ArrayAdapter.
90      //It will update the ListArray and the ListView
91      adapter.add(restaurant);

```

# Controller

- Line 91 - The adapter will add the record to the *ArrayList* and in turn update the *ListView* as one of the displayed items

```

82      //Create Restaurant Data Model
83      Restaurant restaurant = new Restaurant();
84      //Update the Restaurant Data Model
85      restaurant.setName(nameStr);
86      restaurant.setAddress(addressStr);
87      restaurant.setTelephone(telStr);
88      restaurant.setRestaurantType(restType);
89      //Pass the record to ArrayAdapter.
90      //It will update the ListArray and the ListView
91      adapter.add(restaurant);

```

# Model

- Example of how a setter function works:

1. In the class Restaurant Model, it has a local variable “**restaurantName**” (**PURPLE** colour) in memory

```
private String restaurantName = "";
```

2. Link an **EditText variable** to the **EditText** view widget that is to input the restaurant name through “**findViewById**” using the “**id**” defined

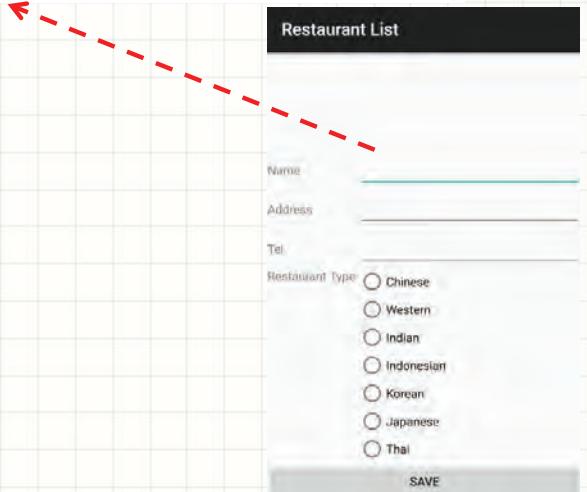
```
private EditText restaurantName;  
restaurantName = findViewById(R.id.restaurant_name);
```

# Model

- Example of how a setter function works:

3. The restaurant name entered by the user can be read from the EditText view using *getText* method from **EditText variable** `restaurantName` and save it to a local String variable “`nameStr`”.

```
String nameStr = restaurantName.getText().toString();
```



# Model

- Example of how a setter function works:

4. Create a Restaurant model, “`Restaurant`”. Pass the “`nameStr`” read from EditText view to model by calling `setName` method (see the RED arrow)

```
Restaurant restaurant = new Restaurant();
restaurant.setName(nameStr);
```



```
public void setName(String restaurantName) {
    this.restaurantName = restaurantName;
}
```

# Model

- Example of how a setter function works:
  5. Data in “restaurantName” (BLACK colour) will then be saved to Model’s local variable “restaurantName” (PURPLE colour)

```
public void setName(String restaurantName) {
    this.restaurantName = restaurantName;
}
```

# Model

- Example of how a getter function works:

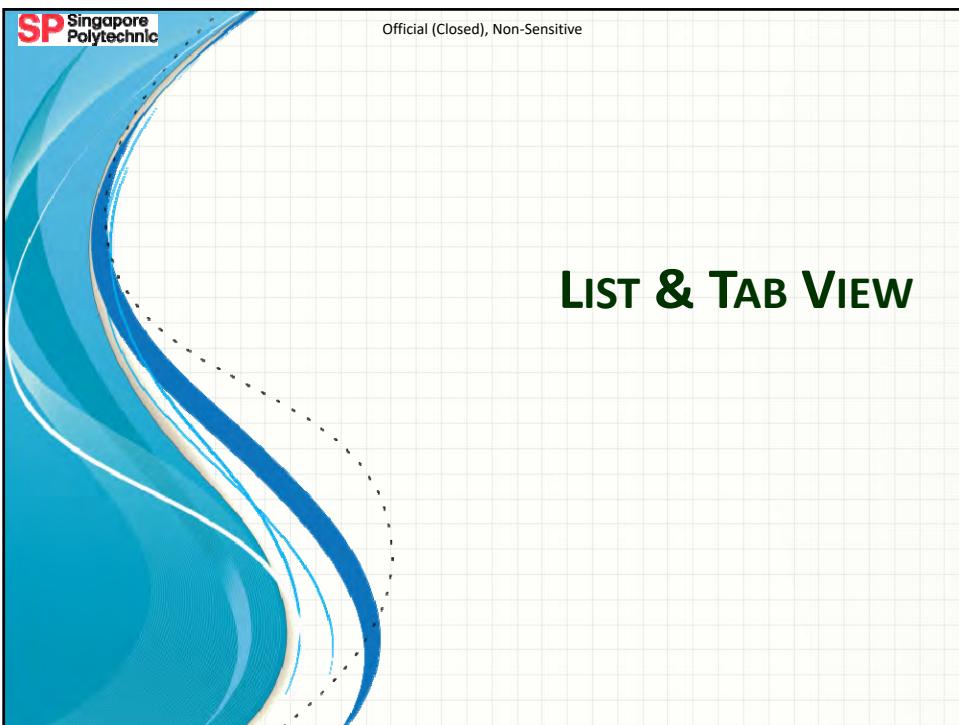
When the getter method *getName* is called, the data (i.e. the data previously stored by *setName* method) in the “restaurantName” (PURPLE colour) variable in Restaurant model will be returned.

```
public String getName() {
    return (restaurantName);
}
```

Calling the getter method through a Restaurant model.

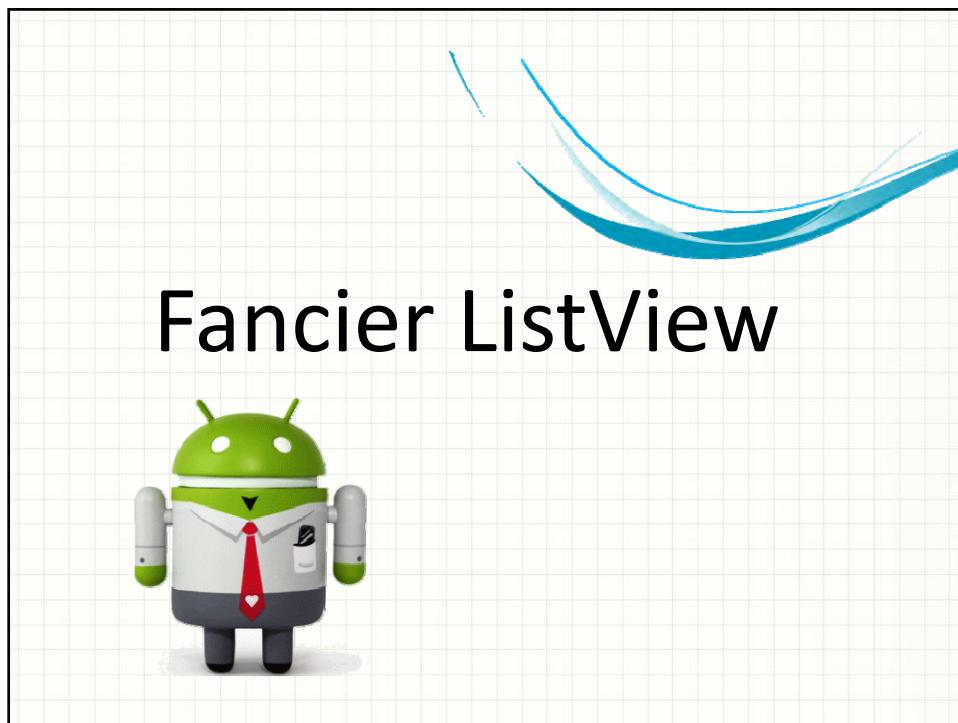
```
Restaurant restaurant = new Restaurant();
restaurant.getName();
```

```
public String toString() {
    return (getName());
}
```



The slide has a blue wavy sidebar on the left. The main content area is titled "Today's Overview" in large black font. To the right of the title are two green rounded rectangles containing numbers 1 and 2 respectively. To the right of each number is a bulleted list item.

| Step | Description                 |
|------|-----------------------------|
| 1    | • Creating Fancier ListView |
| 2    | • Creating Tabs             |

A slide comparing two versions of a "Restaurant List" screen. The left version shows a simple list with a yellow highlight on the first item. The right version shows a more complex row with an image icon, the restaurant name, and address. A red arrow points from the left screen to the right screen.

Fancier ListView

- Instead of a simple row display in a *ListView*, it can be customized to enhance the look. As shown below, the new row display comprises restaurant name and address of the restaurant, plus an image icon

| Restaurant List  |                               |
|------------------|-------------------------------|
| Jade             | Thai Express                  |
| Name:            | Thai Express                  |
| Address:         | JCube                         |
| Tel:             | 62345678                      |
| Restaurant Type: | <input type="radio"/> Chinese |

| Restaurant List  |                               |
|------------------|-------------------------------|
|                  | Thai Express                  |
| Thai Express     |                               |
| Name:            | Thai Express                  |
| Address:         | JCube                         |
| Tel:             | 62345678                      |
| Restaurant Type: | <input type="radio"/> Chinese |

## Fancier ListView



- Let's check what we need to modify from the previous exercise to incorporate the new row display?
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do any thing new?

## Fancier ListView



**Model** → **NO**

No change in *ArrayList* with Restaurant Data Model

**View** → **YES**

**New row layout** in *ListView* is needed. We will create a *row.xml* layout

**Controller** → **YES**

**New customized Adapter** to inflate the new row display in the *ListView*. We create a subclass *RestaurantAdapter* of *ArrayAdapter* to control the row design and populate row data from the *ArrayList*

# View - New Row Layout

## Row Layout

- The new row layout *row.xml* will contain
  - one *ImageView* widget for image icon
  - two *TextView* widgets for name and address



```
<LinearLayout android:orientation="horizontal">
    <ImageView android:id="@+id/icon"/>
    <LinearLayout android:orientation="vertical">
        <TextView android:id="@+id/restName"/>
        <TextView android:id="@+id/restAddr"/>
    </LinearLayout>
</LinearLayout>
```

Basic skeleton layout for *row.xml*

## Row Layout

- *row.xml* Listing

```
1   <?xml version="1.0" encoding="utf-8" ?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       android:layout_width="match_parent"
5       android:layout_height="wrap_content"
6       android:orientation="horizontal">
7
8       <ImageView
9           android:id="@+id/icon"
10          android:layout_width="wrap_content"
11          android:layout_height="wrap_content"
12          android:layout_weight="1"
13          app:srcCompat="@android:color/holo_blue_dark" />
14
15       <LinearLayout
16           android:layout_width="match_parent"
17           android:layout_height="wrap_content"
18           android:layout_weight="1"
19           android:orientation="vertical">
20
21           <TextView
22               android:id="@+id/restName"
23               android:layout_width="match_parent"
24               android:layout_height="wrap_content" />
25
26           <TextView
27               android:id="@+id/restAddr"
28               android:layout_width="match_parent"
29               android:layout_height="wrap_content" />
30
31     </LinearLayout>
32
33 </LinearLayout>
```

## It's Quiz Time!

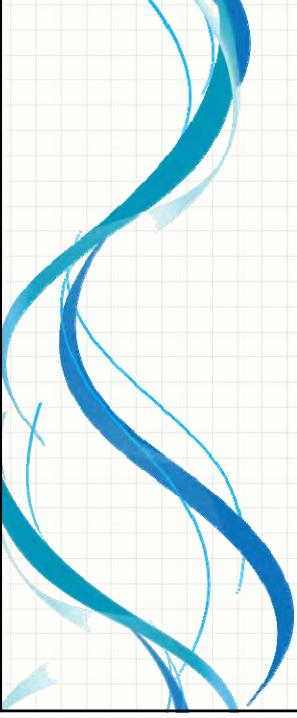
1. In *row.xml* file, the type for the widget with id *restName* is \_\_\_\_\_ . In *main.xml* file, the type for widget with id *restaurant\_name* is \_\_\_\_\_

- A. EditText, EditText
- B. EditText, TextView
- C. TextView, EditText
- D. TextView, TextView

2. Are the display results of the above two widgets the same?

- A. Yes
- B. No

3. What's the difference between the two widgets?



# Controller – Customized Adapter

## Customized Adapter

- In Practical 1, you have learnt that a *ListView* inflates its rows with the *ArrayList* of Restaurant Data Model through *ArrayAdapter*.
- New Customized *Adapter* (a sub-class of  *ArrayAdapter* class) is implemented to incorporate the *row.xml* layout with the *ArrayList*.

```
private List<Restaurant> model = new ArrayList<>();  
  
class RestaurantAdapter extends ArrayAdapter<Restaurant> {  
    RestaurantAdapter() {  
        super(RestaurantList.this, R.layout.row, model);  
    }  
}
```

Compare with `list = findViewById(R.id.restaurants);`  
List view item `adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, model);`  
display in lab 1.

## Customized Adapter

- `getView()` method is responsible for creating the individual rows of *ListView*
- The method determines the layout and the data assignment of the row

```
public View getView(int position, View convertView, ViewGroup parent) {  
    View row = convertView;  
    RestaurantHolder holder = null;  
  
    if (row == null) {  
        LayoutInflater inflater = getLayoutInflater();  
  
        row = inflater.inflate(R.layout.row, parent, false);  
        holder = new RestaurantHolder(row);  
        row.setTag(holder);  
    } else {  
        holder = (RestaurantHolder) row.getTag();  
    }  
    holder.populateFrom(model.get(position));  
  
    return (row);  
}
```

<https://developer.android.com/reference/android/widget/ArrayAdapter>

## Customized Adapter

- If user scrolls the list, certain rows will not be visible anymore in *ListView*
- Android recycles rows which are not displayed anymore and allow these rows to be reused
- A performance optimized adapter assigns the new content to the recycled row. Only updating the content of the row avoids loading the *row.xml* XML layout

## Customized Adapter

- If *ListView* has a recycled row, it passes these rows to *getView()* method as *convertView* parameter
- *convertView* may be **NULL** if there is no row to recycle

```
public View getView(int position, View convertView, ViewGroup parent) {  
    View row = convertView;  
    RestaurantHolder holder = null;  
  
    if (row == null) {  
        LayoutInflator inflater = getLayoutInflater(); →  
        row = inflater.inflate(R.layout.row, parent, false);  
        holder = new RestaurantHolder(row);  
        row.setTag(holder);  
    } else {  
        holder = (RestaurantHolder) row.getTag(); →  
    }  
    holder.populateFrom(model.get(position));  
  
    return (row);  
}
```

No recycled row. Then load the row.xml layout, get ready to be filled with restaurant info.

Recycled row available. Then just refill in the restaurant info to the recycled row, no need to load the row.xml layout.

## View Holder

- The **findViewById()** method is an “expensive” operation, therefore a “View Holder” (*RestaurantHolder*) is created in *Adapter* to hold references to the widgets (**restName**, **restAddr** and **icon**) in the row layout
- This is faster then the repetitive call of the *findViewById()* method

## View Holder

- This “View Holder” is attached to the row via `setTag()` method. If row is recycled, the “View Holder” can be called via `getTag()` method

```
public View getView(int position, View convertView, ViewGroup parent) {
    View row = convertView;
    RestaurantHolder holder = null;

    if (row == null) {
        LayoutInflator inflater = getLayoutInflater();

        row = inflater.inflate(R.layout.row, parent, false);
        holder = new RestaurantHolder(row);
        row.setTag(holder);
    } else {
        holder = (RestaurantHolder) row.getTag();
    }
    holder.populateFrom(model.get(position));

    return (row);
}
```

## View Holder

- RestaurantHolder Class Listing*

```
static class RestaurantHolder {
    private TextView restName = null;
    private TextView addr = null;
    private ImageView icon = null;

    RestaurantHolder(View row) {
        restName = row.findViewById(R.id.restName);
        addr = row.findViewById(R.id.restAddr);
        icon = row.findViewById(R.id.icon);
    }

    void populateFrom(Restaurant r) {
        restName.setText(r.getName());
        addr.setText(r.getAddress() + ", " + r.getTelephone());

        if (r.getRestaurantType().equals("Chinese")) {
            icon.setImageResource(R.drawable.ball_red);
        } else if (r.getRestaurantType().equals("Western")) {
            icon.setImageResource(R.drawable.ball_yellow);
        } else {
            icon.setImageResource(R.drawable.ball_green);
        }
    }
}
```

display corresponding icon based on chosen restaurant's type.

## Binding

- With the customized adapter (*RestaurantAdapter*) ready, it is time to bind the *ListView* to new *ArrayAdapter* -> *RestaurantAdapter* using *setAdapter* method

```
list = findViewById(R.id.restaurants);
adapter = new RestaurantAdapter();
list.setAdapter(adapter);
```

## It's Quiz Time!

- In the code segment below, which parameter of parent class *ArrayAdapter*'s constructor decides the display format of the *Restaurant* object info according to *row.xml* layout?

```
class RestaurantAdapter extends ArrayAdapter<Restaurant> {
    RestaurantAdapter() {
        super(RestaurantList.this, R.layout.row, model);
    }
}
```

- The 1st parameter
- The 2nd parameter
- The 3rd parameter
- None of these

ArrayAdapter  
**public ArrayAdapter (Context context, int resource, T[] objects)**  
Parameters  
**context** Context: The current context. This value must never be null.  
**resource** int: The resource ID for a layout file containing a TextView to use when instantiating views.  
**objects** T: The objects to represent in the ListView. This value must never be null.

## It's Quiz Time!

2. Refer to the partial code listed here, restaurant info (name, address and icon) to be displayed in the list view according to the layout in row.xml is *directly* provided by \_\_\_\_\_.

- A. RestaurantHolder object holder
- B. Restaurant object
- C. EditText widget in UI
- D. None of these

```
static class RestaurantHolder {  
    private TextView restName = null;  
    private TextView addr = null;  
    private ImageView icon = null;  
  
    RestaurantHolder(View row) {  
        restName = row.findViewById(R.id.restName);  
        addr = row.findViewById(R.id.restAddr);  
        icon = row.findViewById(R.id.icon);  
    }  
  
    public View getView(int position, View convertView, ViewGroup parent) {  
        View row = convertView;  
        RestaurantHolder holder = null;  
  
        if (row == null) {  
            LayoutInflater inflater = getLayoutInflater();  
  
            row = inflater.inflate(R.layout.row, parent, false);  
            holder = new RestaurantHolder(row);  
            row.setTag(holder);  
        } else {  
            holder = (RestaurantHolder) row.getTag();  
        }  
        holder.populateFrom(model.get(position));  
  
        return (row);  
    }  
}
```

## It's Quiz Time!

3. Refer to the code listed here, restaurant info (name, address) held by **RestaurantHolder object** which is to be displayed in the list view according to the layout in row.xml is *directly* captured from \_\_\_\_\_.

- A. Restaurant object
- B. EditText widget in UI
- C. Resources in res folder
- D. None of these

```
static class RestaurantHolder {  
    private TextView restName = null;  
    private TextView addr = null;  
    private ImageView icon = null;  
  
    RestaurantHolder(View row) {  
        restName = row.findViewById(R.id.restName);  
        addr = row.findViewById(R.id.restAddr);  
        icon = row.findViewById(R.id.icon);  
    }  
  
    void populateFrom(Restaurant r) {  
        restName.setText(r.getName());  
        addr.setText(r.getAddress() + ", " + r.getTelephone());  
  
        if (r.getRestaurantType().equals("Chinese")) {  
            icon.setImageResource(R.drawable.ball_red);  
        } else if (r.getRestaurantType().equals("Western")) {  
            icon.setImageResource(R.drawable.ball_yellow);  
        } else {  
            icon.setImageResource(R.drawable.ball_green);  
        }  
    }  
}
```

## It's Quiz Time!

4. Refer to the code listed here, restaurant icon info held by RestaurantHolder object which is to be displayed in the list view according to the layout in row.xml is **directly** captured from \_\_\_\_\_.

- A. Restaurant object
- B. EditText widget in UI
- C. Resources in res folder
- D. None of these

```
static class RestaurantHolder {  
    private TextView restName = null;  
    private TextView addr = null;  
    private ImageView icon = null;  
  
    RestaurantHolder(View row) {  
        restName = row.findViewById(R.id.restName);  
        addr = row.findViewById(R.id.restAddr);  
        icon = row.findViewById(R.id.icon);  
    }  
  
    void populateFrom(Restaurant r) {  
        restName.setText(r.getName());  
        addr.setText(r.getAddress() + ", " + r.getTelephone());  
  
        if (r.getRestaurantType().equals("Chinese")) {  
            icon.setImageResource(R.drawable.ball_red);  
        } else if (r.getRestaurantType().equals("Western")) {  
            icon.setImageResource(R.drawable.ball_yellow);  
        } else {  
            icon.setImageResource(R.drawable.ball_green);  
        }  
    }  
}
```

## It's Quiz Time!

5. (Revision) Refer to the code listed here, Restaurant object info(name, address, telephone No.) is captured from \_\_\_\_\_.

- A. EditText widget in UI
- B. Resources in res folder
- C. TextView widget in UI
- D. None of these



```
<EditText  
    android:id="@+id/restaurant_name"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:ems="10"  
    android:inputType="textPersonName"  
    android:maxLength="30" />  
  
35     buttonSave = findViewById(R.id.button_save);  
36     buttonSave.setOnClickListener(onSave);  
  
37     private View.OnClickListener onSave = new View.OnClickListener() {  
38         @Override  
39         public void onClick(View v) {  
40             // To read data from restaurantName EditText  
41             String nameStr = restaurantName.getText().toString();  
42             // To read data from restaurantAddress EditText  
43             String addressStr = restaurantAddress.getText().toString();  
44             // To read data from restaurantTel EditText  
45             String telStr = restaurantTel.getText().toString();  
46             //Create Restaurant Data Model  
47             Restaurant restaurant = new Restaurant();  
48             //Update the Restaurant Data Model  
49             restaurant.setName(nameStr);  
50             restaurant.setAddress(addressStr);  
51             restaurant.setTelephone(telStr);  
52         }  
53     };
```

## It's Quiz Time!

6. (Revision) Refer to the code listed here, Restaurant object's restaurant type info is captured from \_\_\_\_\_.

- A. EditText widget in UI
- B. Resources in res folder
- C. TextView widget in UI
- D. RadioGroup widget in UI

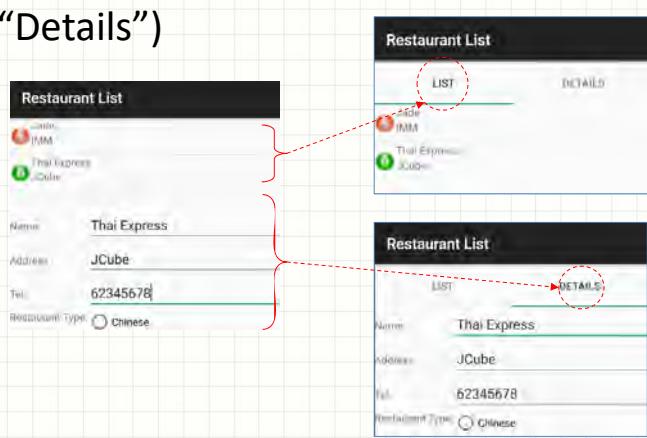
```
buttonSave = findViewById(R.id.button_save);
buttonSave.setOnClickListener(onSave);

private View.OnClickListener onSave = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //To read selection of restaurantTypes RadioGroup
        int radioID = restaurantTypes.getCheckedRadioButtonId();
        if (radioID == R.id.chinese) {
            restType = "Chinese";
        } else
        if (radioID == R.id.western) {
            restType = "Western";
        } else
        if (radioID == R.id.indian) {
            restType = "Indian";
        } else
        if (radioID == R.id.indonesian) {
            restType = "Indonesian";
        } else
        if (radioID == R.id.korean) {
            restType = "Korean";
        } else
        if (radioID == R.id.japanese) {
            restType = "Japanese";
        } else
        if (radioID == R.id.thai) {
            restType = "Thai";
        }
    }
}
```

## SPLITTING THE VIEWS

## Tabs

- In the second part of Practical 2 exercise, the *ListView* and Restaurant Form are split and displayed into two separated tabs (“List” and “Details”)



## Tabs

- Let's check what we need to modify from the previous exercise to split the views?
- Model** - Any change in Data Model?
- View** - Do you need to modify any of the user interface view?
- Controller** - Do you need to tell the Controller to do any thing new?

## Tabs

Model → NO

No change in *ArrayList* with Restaurant Data Model

View → YES

Tab layout is needed. We will modify the *main.xml* layout to use *TabHost*, *TabWidget* and *FrameLayout* to create the new view

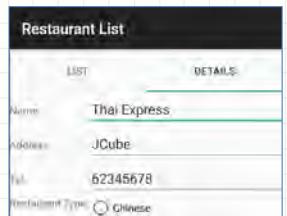
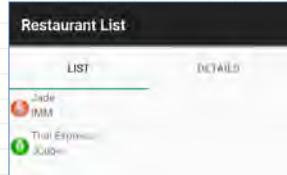
Controller → YES

Create tabs for “List” and “Details” UI views

UI View -  
New Main  
Layout

## Tab View

- Update the *main.xml* layout with *TabHost*, *TabWidget* and *FrameLayout* for the new layout



```
<TabHost>
<LinearLayout android:orientation="vertical" >
<TabWidget />
<FrameLayout>

    <LinearLayout android:id="@+id/restaurants_tab">
        <ListView android:id="@+id/restaurants" />
    </LinearLayout>

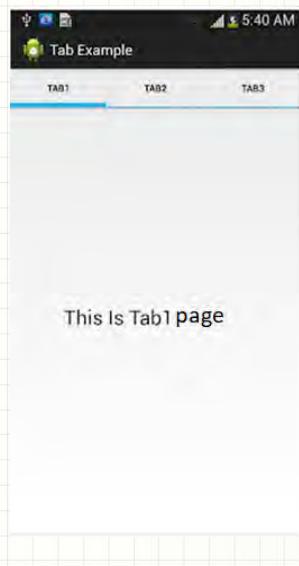
    <LinearLayout android:id="@+id/details_tab">
        <TableLayout android:id="@+id/details" >
            <TableRow>
            </TableRow>
            .....
        </TableLayout>
    </LinearLayout>
</FrameLayout>
</LinearLayout>
</TabHost>
```

Basic Skeleton of main.xml

## Tab View

### TabHost

- It provides a nice way to present multiple thing on a Single Screen.
- It is a **container** for a tabbed window view
- This object **holds two children**:
  - a set of **tab labels** that the user clicks to select a specific tab, and
  - a **FrameLayout object** that displays the contents of that page. It contains the UI that will be shown when user selects/clicks particular tab.



## Tab View

### TabHost

- *TabHost* is used to add labels, add the callback handler, and manage callbacks. For instance, switching the displayed page.
- It contains one or more tab (TabHost.TabSpec).

## Tab View

### TabWidget

- Displays a list of **tab labels** representing each page in the TabHost's tab collection.
- When the user selects a tab, the *TabWidget* sends a message to the parent container, *TabHost*, to tell it to switch the displayed page.

## Main Layout

- *main.xml* Listing (partial)

```
13 <TabHost>
14     android:id="@+id/tabHost"
15     android:layout_width="match_parent"
16     android:layout_height="match_parent">
17
18     <LinearLayout
19         android:layout_width="match_parent"
20         android:layout_height="match_parent"
21         android:orientation="vertical">
22
23         <TabWidget
24             android:id="@+android:id/tabs"
25             android:layout_width="match_parent"
26             android:layout_height="wrap_content" />
27
28         <FrameLayout
29             android:id="@+android:id/tabcontent"
30             android:layout_width="match_parent"
31             android:layout_height="match_parent">
```

# Controller – Setting up Tabs View

## Controller

- Create the two tabs and load with different UI View content defined in *FrameLayout*

- restaurants (*ListView*) for ‘List’ tab

```
//Tab 1
TabHost.TabSpec spec = host.newTabSpec("List");
spec.setContent(R.id.restaurants_tab);
spec.setIndicator("List");
host.addTab(spec);
```

- restaurantDetails (*TableLayout*) for ‘Details’ tab

```
//Tab 2
spec = host.newTabSpec("Details");
spec.setContent(R.id.details_tab);
spec.setIndicator("Details");
host.addTab(spec);
```

```
<FrameLayout
    android:id="@+id/tabcontent"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:id="@+id/restaurants_tab"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ListView
            android:id="@+id/restaurants"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/details_tab"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

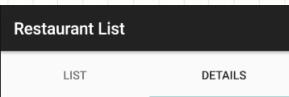
        <TableLayout...>
        <Button...>
    </LinearLayout>
</FrameLayout>
```

## Controller

- Setup the title display (“LIST” and “DETAILS”) of each tab using *setIndicator* method
- Add the new tab to the *TabHost* using *addTab* method

```
spec.setIndicator("List");
host.addTab(spec);

spec.setIndicator("Details");
host.addTab(spec);
```



## Controller

### TabHost.TabSpec

- A tab has a tab indicator, content, and a tag that is used to keep track of it.
- For the **tab indicator**, your choices are:
  - 1) set a **label**
  - 2) set a **label** and an **icon**

|                                 |                                                                                                                   |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <a href="#">TabHost.TabSpec</a> | <a href="#">setIndicator(CharSequence label)</a><br>Specify a label as the tab indicator.                         |
| <a href="#">TabHost.TabSpec</a> | <a href="#">setIndicator(CharSequence label, Drawable icon)</a><br>Specify a label and icon as the tab indicator. |

## Controller

### TabHost.TabSpec

- For the **tab content**, your choices are:
  - 1) the **id** of a View
  - 2) a **TabContentFactory** that creates the View content.
  - 3) an **Intent** that launches an Activity.

|                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------|
| <a href="#">setContent(Intent intent)</a><br>Specify an intent to use to launch an activity as the tab content. |
|-----------------------------------------------------------------------------------------------------------------|

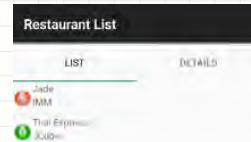
| [setContent\(TabHost.TabContentFactory contentFactory\)](#) Specify a [TabHost.TabContentFactory](#) to use to create the content of the tab. |
| [setContent\(int viewId\)](#) Specify the id of the view that should be used as the content of the tab. |

## Controller

- After setting up the UI View to link to the Controller, `setCurrentTab(int)` method can be used to control the UI View to be displayed
  - `host.setCurrentTab(0)` to show *ListView*
  - `host.setCurrentTab(1)` to show the *Detail form*

## Controller – Detecting List Item Click

## List Click



- Let's check what do we need to modify from the previous exercise to detect list item click?
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do any thing new?

## List Click

- Model** - NO
- View** - NO
- Controller** - YES

Similar to Button 'click', you need to implement an on item event listener to the *ListView*. When 'clicking' on the list item, the Controller will activate the listener (*AdapterView* on item click listener) to handle the event.

## Controller

- Implement the Controller to capture the event generated by *ListView* when a row is selected using *setOnItemClickListener* method and pass the event to *onListClick* (*ArrayList* *onItemClick* Listener)

```
list = findViewById(R.id.restaurants);
adapter = new RestaurantAdapter();
list.setAdapter(adapter);

list.setOnItemClickListener(onListClick);

private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        private View.OnClickListener onSave = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                buttonSave.setOnClickListener(onSave);
            }
        };
    }
}
```

Compared with  
button click handler

## Controller

- When a row is selected, the *ListView* selected event will be captured by the *AdapterView.OnItemClickListener* class and handle by *onItemClick()* method for the necessary action. In this case, it is to get the restaurant data from the *ArrayList* Model to memory (restaurant Model) and transfer to the widgets (name, address, telephone and restaurantType) on the detail form

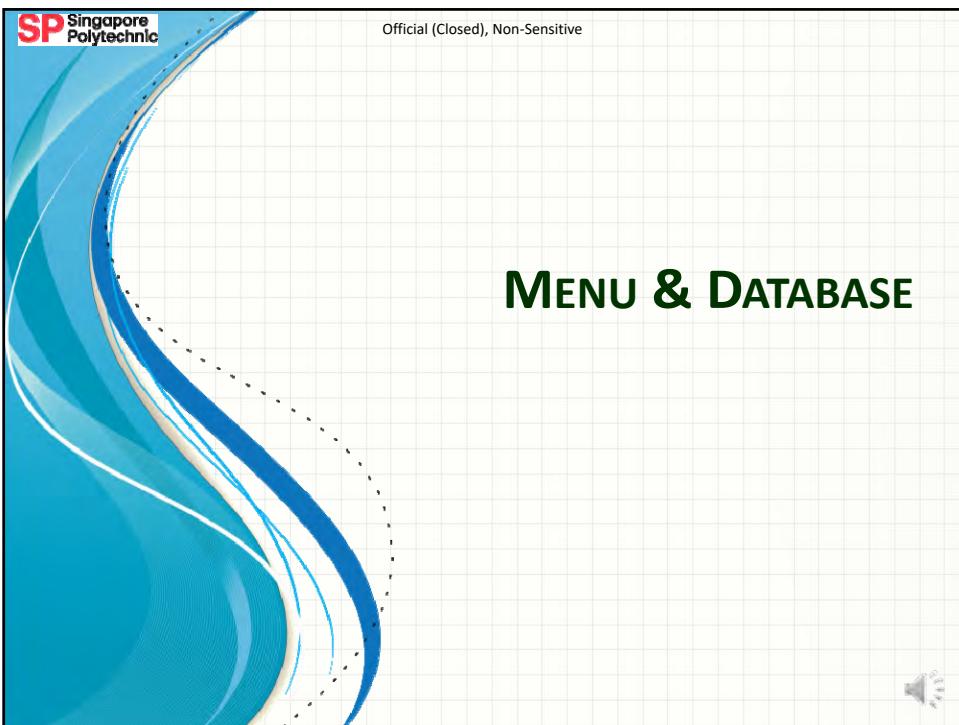
```
private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
```

# Controller

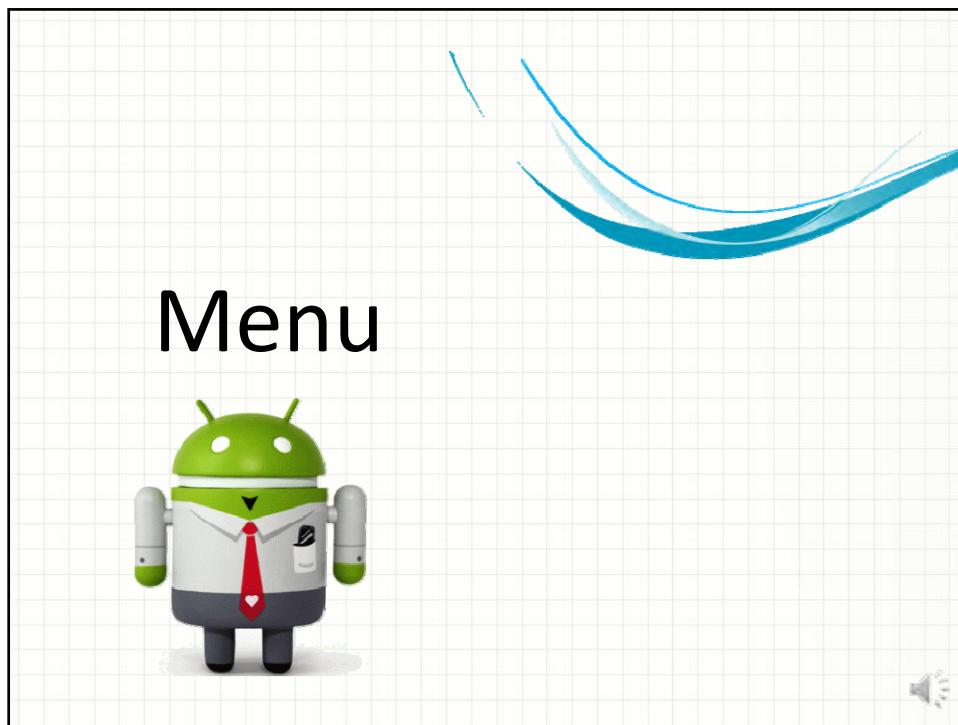
- Respond to row selected event on *ListView*

```
private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {  
  
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
        Restaurant r = model.get(position); ← Get data from ArrayList Model to restaurant Model  
        restaurantName.setText(r.getName());  
        restaurantAddress.setText(r.getAddress());  
        restaurantTel.setText(r.getTelephone());  
  
        if (r.getRestaurantType().equals("Chinese")) {  
            restaurantTypes.check(R.id.chinese);  
        } else if (r.getRestaurantType().equals("Western")) {  
            restaurantTypes.check(R.id.western);  
        } else if (r.getRestaurantType().equals("Indian")) {  
            restaurantTypes.check(R.id.indian);  
        } else if (r.getRestaurantType().equals("Indonesia")) {  
            restaurantTypes.check(R.id.indonesian);  
        } else if (r.getRestaurantType().equals("Korean")) {  
            restaurantTypes.check(R.id.korean);  
        } else if (r.getRestaurantType().equals("Japanese")) {  
            restaurantTypes.check(R.id.japanese);  
        } else {  
            restaurantTypes.check(R.id.thai);  
        }  
  
        host.setCurrentTab(1); ← Switch the current UI display to "Details" tab  
    };  
};
```

Transfer individual data element to widgets on UI View



The slide has a blue vertical bar on the left. The title "Today's Overview" is at the top. Below it, two green rounded rectangles contain the numbers "1" and "2" respectively. To the right of each number is a bullet point: "• Menu" next to "1" and "• Database" next to "2". The background is a grid pattern. A small speaker icon is in the bottom right corner.



A vertical sequence of three screenshots from a mobile application. The first screenshot shows a navigation bar with 'Restaurant List' at the top, followed by 'LIST' and 'DETAILS' buttons. A red dashed box highlights the three-dot menu icon. The second screenshot shows the same navigation bar, but the menu is open, revealing options like 'About'. The 'About' option is highlighted with a red box. The third screenshot shows the menu closed again, and the 'DETAILS' screen is displayed, showing fields for 'Name', 'Address', 'Type', and a list of 'Restaurant Types' with radio buttons. The 'Western' type is selected. Red dashed boxes highlight the menu icon in the first screenshot, the open menu in the second, and the 'DETAILS' screen in the third.

## Menu

- Let's check what do we need to modify from the previous exercise to incorporate the new MENU display?
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do any thing new?



## Menu

**Model** - NO

**View** - YES

Has to add a MENU with the label 'About'. The menu layout file named 'option.xml' will be created.

**Controller** – YES

We will need to update the Controller to link to the menu item 'About' and implement the action when it is selected



# UI View - Menu



## View

### Menu

- The **MENU** is designed with the ‘option.xml’ layout.
- Each menu item option is an **item** in the MENU layout.
- For example, the menu item with the label ‘About’ with the id ‘*about*’ is added to the MENU
- When **MENU** on the device is pressed, the menu item ‘About’ will be shown.

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

    <item
        android:id="@+id/about"
        android:title="About" />
</menu>
```



# Controller – Menu



## Menu

- When the **MENU** is pressed, by default, the Controller will trigger the **onCreateOptionsMenu(Menu)** method and inflate the layout defined by *option.xml* for current UI View

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.option, menu);  
    return super.onCreateOptionsMenu(menu);  
}
```

Restaurant List



## It's Quiz Time!

1. When the MENU is pressed, which method will be triggered?

\_\_\_\_\_



## It's Quiz Time! (Solution)

1. When the MENU is pressed, which method will be triggered?

-> ***onCreateOptionsMenu(Menu)***



# UI View – Menu Item Detection



## MENU ITEM

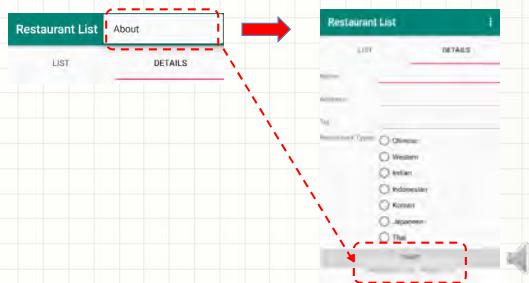
- Let's check what do we need to modify from the previous exercise to detect MENU item selection?
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do any thing new?



## MENU ITEM

- Model - NO
- View - NO
- Controller – YES

`onOptionsItemSelected(MenuItem item)` method is added to the Controller to handle the option item select event



## MENU ITEM

- When an menu option item is selected, the Controller will capture the selection event and trigger the `onOptionsItemSelected(MenuItem)` method to take action. In this case, a **Toast** message box is used to acknowledge the event triggered.
- The `getItemId()` method return the **id** of the menu option item selected.

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    if (item.getItemId() == R.id.about) {  
        Toast.makeText(context, text: "Restaurant List - version 1.0", Toast.LENGTH_LONG).show();  
    }  
    return super.onOptionsItemSelected(item);  
}
```

# **DATABASE MODEL**



## Database

- The restaurant Model created so far to hold the restaurant data is temporary! When user presses the BACK button and launch the Restaurant List app again, the data in restaurants list entered will be gone. This is because the Model uses memory to store its information

Volatile MEMORY

Restaurant 1

Restaurant 2

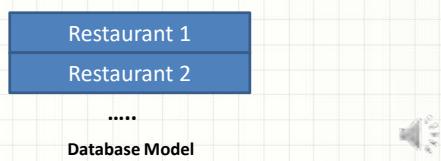
.....

ArrayList Model



## Database

- What happen if the restaurant data needs to stay and non-volatile?
- In second part of Practical 3 exercise, a non-volatile data Model (Database) is used to replace the volatile data Model (ArrayList)
- The restaurant data is saved as a local file using SQLite database Non-volatile MEMORY



## Database

- Let's check what do we need to modify from the previous exercise to use SQLite database?
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do any thing new?

## Database

### Model - YES

Due to a total changed in Data Model, the *ArrayList* is replaced by *restaurantlist.db* database with *restaurants\_table* as Data Model. The *Restaurant.java* file is deleted and replaced by *RestaurantHelper.java* (a sub-class of *SQLiteOpenHelper* )



## Database

### View - NO

### Controller - YES

When using memory as a temporary data storage, *ArrayList* and *ArrayAdapter* are used.

When data is stored as a record file in *restaurants\_table*, *Cursor* and *CursorAdapter* will be used for handling database Model and *ListView*



## It's Quiz Time!

1. Why do we need a database for mobile app?

---

2. Volatile memory is also temporary memory – True or False?

3. Select the correct aspects of volatile and non-volatile memory

- Volatile data model is ArrayList
- Non-Volatile data model is Database
- ArrayList and ArrayAdapter are used for non-volatile memory
- restaurants\_table, Cursor and CursorAdapter are used for volatile memory



## It's Quiz Time! (Solution)

1. Why do we need a database for mobile app?

*In certain situation, you need your mobile app data to stay and non-volatile.*

2. Volatile memory is also temporary memory – True

3. Select the correct aspects of volatile and non-volatile memory

- ✓ **Volatile data model is ArrayList**
- ✓ **Non-Volatile data model is Database**
- ✗ ArrayList and ArrayAdapter are used for non-volatile memory
- ✗ restaurants\_table, Cursor and CursorAdapter are used for volatile memory



# Model - Database Model



## Model

- The *RestaurantHelper.java* “helper” (*SQLiteOpenHelper* sub-class) is created to have access to database Model

```
class RestaurantHelper extends SQLiteOpenHelper {
```

- It provides methods for creating and opening **database Model**, and inserting, updating and reading data record from **table Model**



# Model

## RestaurantHelper

- A **table model** named *restaurants\_table* is created with the necessary data elements structure

```
@Override  
public void onCreate(SQLiteDatabase db) {  
    // Will be called once when the database is not created  
    db.execSQL("CREATE TABLE restaurants_table (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +  
              "restaurantName TEXT, restaurantAddress TEXT, restaurantTel TEXT, restaurantType TEXT);");  
}
```

- Table model data structure

| Field Name        | Type    | Key     |                           |
|-------------------|---------|---------|---------------------------|
| _id               | INTEGER | PRIMARY | Unique ID for each record |
| restaurantName    | TEXT    |         |                           |
| restaurantAddress | TEXT    |         |                           |
| restaurantTel     | TEXT    |         |                           |
| restaurantType    | TEXT    |         |                           |



# Model

## RestaurantHelper

- A **table model** is a **collection of records**. Each record contains data of the fields as defined in the table model data structure.
- An example of a restaurant record:
  - \_id : 1
  - restaurantName : Jade
  - restaurantAddress : IMM
  - restaurantTel : 61234567
  - restaurantType : Chinese
- A record can be inserted into, updated, deleted or read from the table model.



# Model

## RestaurantHelper

- `SQLiteOpenHelper` provides methods `getReadableDatabase()` and `getWritableDatabase()` to get access to `SQLiteDatabase` objects; either in read or write mode.
- `getAll()` read all records from the **table model**, `restaurants_table` by executing a SQL query against the `SQLiteDatabase database model` return by `getReadableDatabase()`.

```
/* Read all records from restaurants_table */
public Cursor getAll() {
    return [getReadableDatabase()].rawQuery(
        "SELECT _id, restaurantName, restaurantAddress, restaurantTel, restaurantType " +
        "FROM restaurants_table ORDER BY restaurantName", null);
}
```



# Model

## RestaurantHelper

- The `insert(String restaurantName, String restaurantAddress, String restaurantTel, String restaurantType)` method is responsible for inserting a record into the **table model**, `restaurants_table` with the data that are passed as parameters to the method against the `SQLiteDatabase database model` return by `getWritableDatabase()`.

```
/* Write a record into restaurants_table */
public void insert(String restaurantName, String restaurantAddress, String restaurantTel,
    String restaurantType) {
    ContentValues cv = new ContentValues();

    cv.put("restaurantName", restaurantName);
    cv.put("restaurantAddress", restaurantAddress);
    cv.put("restaurantTel", restaurantTel);
    cv.put("restaurantType", restaurantType);

    [getWritableDatabase()].insert("restaurants_table", "restaurantName", cv);
}
```



# Controller – Setting up



## Controller

- The Adapter to handle the Database Model and *ListView* is replaced by the *CursorAdapter*
- A **cursor** is a collection of records read from the table Model.
- The *CursorAdapter* creates a *View* for each row in a Cursor where each row correspond to a record read from the table Model
- A *CursorAdapter* does not use *getView()* (used in the ArrayAdapter), but rather the *newView()* and *bindView()* methods.

```
class RestaurantAdapter extends CursorAdapter {  
    RestaurantAdapter(Context context, Cursor cursor, int flags) {  
        super(context, cursor, flags);  
    }  
  
    @Override  
    public void bindView(View view, Context context, Cursor cursor) {  
        RestaurantHolder holder = (RestaurantHolder) view.getTag();  
        holder.populateFrom(cursor, helper);  
    }  
  
    @Override  
    public View newView(Context context, Cursor cursor, ViewGroup parent) {  
        LayoutInflater inflater = getLayoutInflater();  
        View row = inflater.inflate(R.layout.row, parent, false);  
        RestaurantHolder holder = new RestaurantHolder(row);  
        row.setTag(holder);  
        return (row);  
    }  
}
```

## Controller CursorAdapter

- The newView() method handles inflating new row in the UI View

```
@Override  
public View newView(Context context, Cursor cursor, ViewGroup parent) {  
    LayoutInflater inflater = getLayoutInflater();  
    View row = inflater.inflate(R.layout.row, parent, false);  
    RestaurantHolder holder = new RestaurantHolder(row);  
    row.setTag(holder);  
    return (row);  
}
```

- The bindView() handles recycled row disappears from the UI View

```
@Override  
public void bindView(View view, Context context, Cursor cursor) {  
    RestaurantHolder holder = (RestaurantHolder) view.getTag();  
    holder.populateFrom(cursor, helper);  
}
```



## Controller

- Instead of using *ArrayList* Model to hold the table Model, it is changed to *Cursor*
- The *getAll()* method will return the Cursor (rows of data from table Model)

```
private Cursor model = null;  
  
helper = new RestaurantHelper(this);  
list = (ListView) findViewById(R.id.restaurants);  
model = helper.getAll();  
adapter = new RestaurantAdapter(model);  
list.setAdapter(adapter);
```



## Controller

- When a new record is saved to table Model using *insert()* method, the *Cursor model* need to be updated by re-reading from the table model again and update the *CursorAdapter*.
- Using the *swapCursor()* method of the *CursorAdapter* will update the *View* with the *updated Cursor model*. So doing, the List View will be updated to list the newly inserted record.

```
//Insert record into SQLite table
helper.insert(nameStr, addrStr, telStr, restType);

model = helper.getAll();      //Reload the content of Cursor (record list)
adapter.swapCursor(model);   //Update List View with new record list
```



## It's Quiz Time!

1. SQL command "SELECT \_id FROM restaurants ...." is to read record from table model - True or False?

2. What is the purpose of newView method below?

```
public View newView(Context context, Cursor cursor, ViewGroup parent) {
    LayoutInflator inflater = getLayoutInflater();
    View row = inflater.inflate(R.layout.row, parent, false);
    RestaurantHolder holder = new RestaurantHolder(row);
    row.setTag(holder);
    return (row);
}
```

---

3. Cursor is used to replace ArrayList model and hold Table model - True or False?



## It's Quiz Time! (Solution)

1. SQL command "SELECT \_id FROM restaurants ...." is to read record from table model - **True**

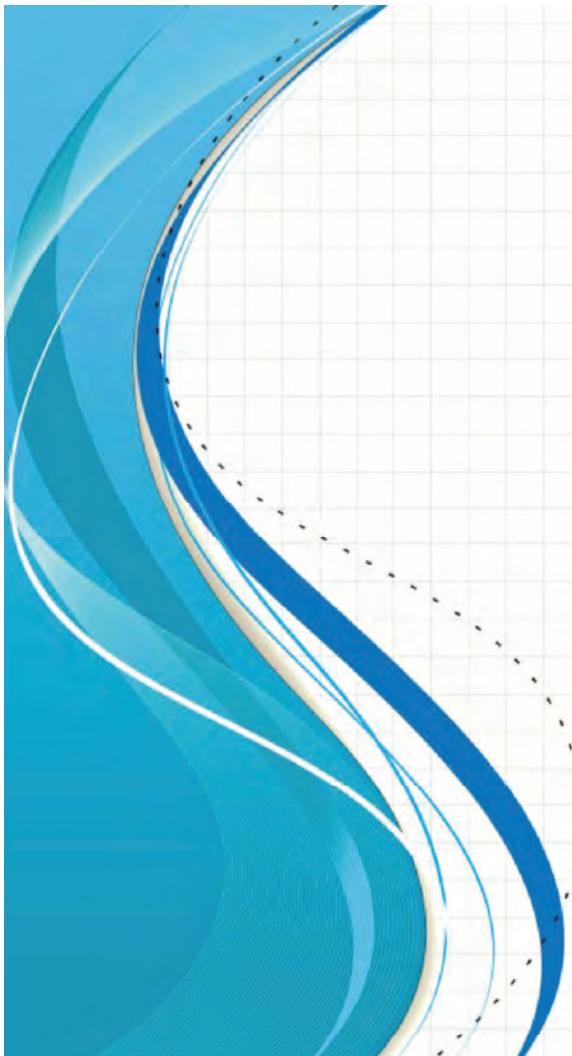
2. What is the purpose of newView method below?

```
public View newView(Context context, Cursor cursor, ViewGroup parent) {  
    LayoutInflator inflater = getLayoutInflator();  
    View row = inflater.inflate(R.layout.row, parent, false);  
    RestaurantHolder holder = new RestaurantHolder(row);  
    row.setTag(holder);  
    return (row);  
}
```

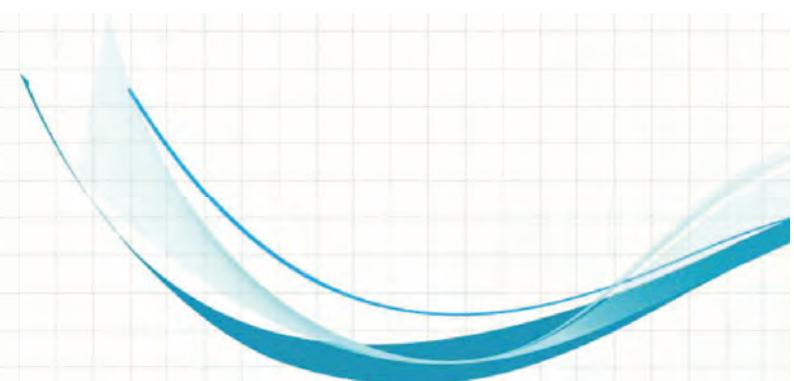
*The newView() method handles inflating new row in the UI View*

3. Cursor is used to replace ArrayList model and hold Table model  
- **True**





## ACTIVITIES



# Activities



# Activity

- In Practical 4 exercise, the *Restaurant List* application will be split into two Activities
  - one *AppCompatActivity* (*RestaurantList*) to handle ‘List’ UI view
  - one *AppCompatActivity* (*DetailForm*) to handle ‘Details’ form UI view



# Activity

- Let’s check what do we need to modify from the previous exercise to split the UI views to two separate *Activities*?
  - ❑ **Model** - Any change in Data Model?
  - ❑ **View** - Do you need to modify any of the user interface view?
  - ❑ **Controller** - Do you need to tell the Controller to do anything new?

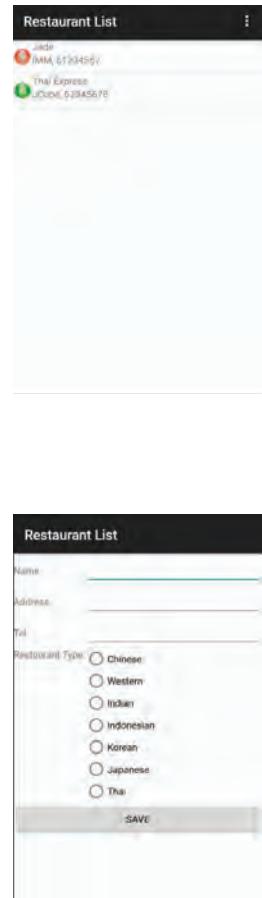
# Activity

❑ Model - NO

❑ View - YES

the UI View will be split into two XML layout files:

- ✓ *main.xml* layout for the ‘List’ UI view
- ✓ *detail\_form.xml* for the ‘Details’ form UI view



# Activity

❑ View - YES

A “Add” menu item added to the Menu for adding new restaurant record to the restaurants table Model. The *option.xml* layout file in the **res/menu** folder will need to be changed to include the new menu item to the **Menu** View.



# Activity

Controller - YES

Now that we split the Restaurant List app to two activities, each activity will have its own Controller:

- ✓ *RestaurantList AppCompatActivity* Controller – to handle the updating of ‘List’ UI view from the *Cursor Model*, calling *DetailForm.java* Activity through Explicit Intent and handle the MENU items selection.
- ✓ *DetailForm AppCompatActivity* Controller – to handle the adding of a new record to the restaurants table Model.

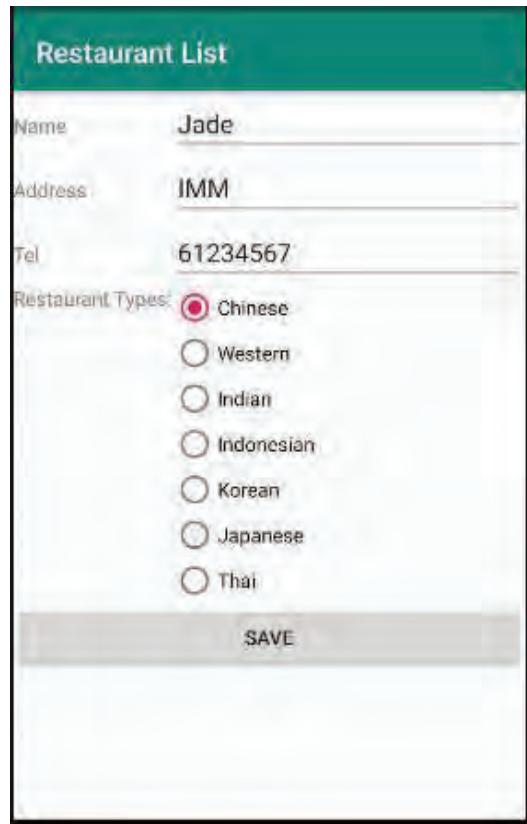
# UI View - Separate Layouts & Option Menu

## View

### 'Details' Form UI View

- The layout for new 'Detail' form activity is taken out from the **original *main.xml*** layout and saved as a separated layout file ***detail\_form.xml*** in **res/layout** folder

Detail Form  
activity in  
*detail\_form.xml*



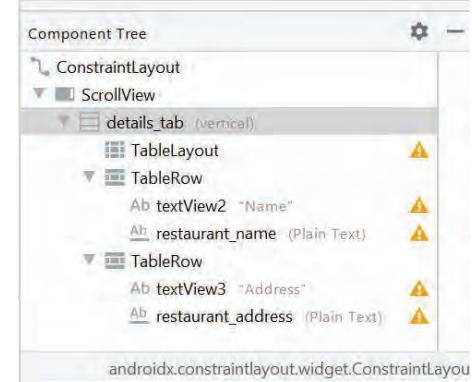
# View detail\_form.xml Listing

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RestaurantList">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:id="@+id/details_tab"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <TableLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:stretchColumns="1">
                <TableRow>
                    <EditText
                        android:id="@+id/restaurant_name"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:ems="10"
                        android:inputType="textPersonName"
                        android:maxLength="30" />
                </TableRow>
                <TableRow>
                    <TextView
                        android:id="@+id/textView"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:text="Name" />
                </TableRow>
            </TableLayout>
        </LinearLayout>
    </ScrollView>
</ConstraintLayout>
```

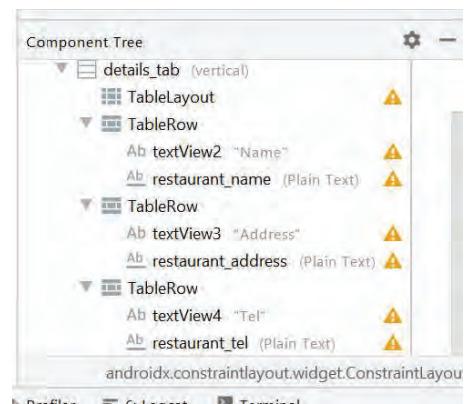


# View detail\_form.xml Listing

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TableRow>
        <TextView
            android:id="@+id/textView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Address" />
        <EditText
            android:id="@+id/restaurant_address"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPostalAddress"
            android:maxLength="60" />
    </TableRow>

    <TableRow>
        <TextView
            android:id="@+id/textView3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Tel" />
        <EditText
            android:id="@+id/restaurant_tel"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="phone"
            android:maxLength="8" />
    </TableRow>
</TableLayout>
```



## View detail\_form.xml Listing

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

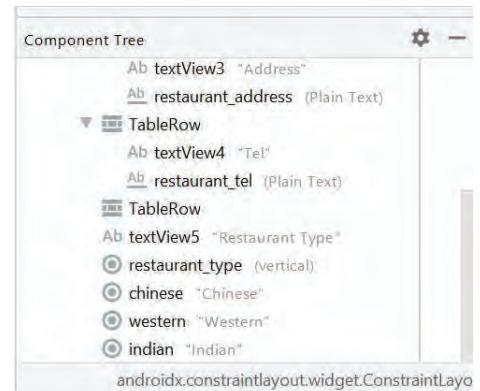
    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Restaurant Types:" />

    <RadioGroup
        android:id="@+id/restaurant_types"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <RadioButton
            android:id="@+id/chinese"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Chinese" />

        <RadioButton
            android:id="@+id/western"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Western" />

        <RadioButton
            android:id="@+id/indian"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Indian" />
    
```



## View detail\_form.xml Listing

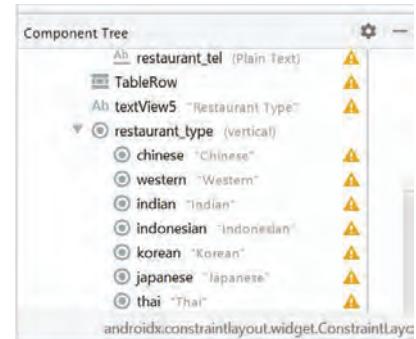
```
<RadioButton
    android:id="@+id/indonesian"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Indonesian" />

<RadioButton
    android:id="@+id/korean"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Korean" />

<RadioButton
    android:id="@+id/japanese"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Japanese" />

<RadioButton
    android:id="@+id/thai"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Thai" />

</RadioGroup>
```

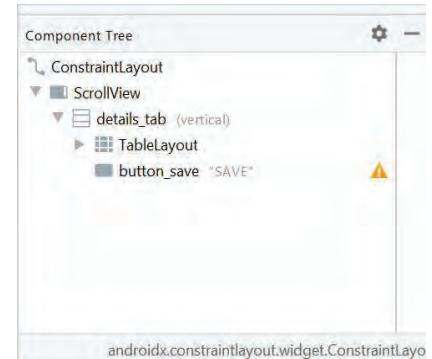


# View detail\_form.xml Listing

```
</TableRow>
</TableLayout>

<Button
    android:id="@+id/button_save"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Save" />
</LinearLayout>
</ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```



# View

## 'Restaurant List' UI View

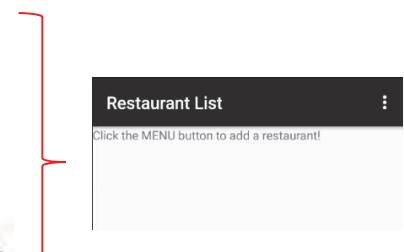
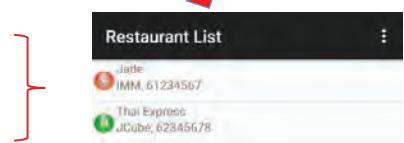
- The *main.xml* layout contains only the *ListView* layout to display **records saved** in the SQLite database and a *TextView* to display an instruction message “**Click the MENU button to add a restaurant!**” when there are **no** restaurant record in the SQLite database.

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...
    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/empty"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Click the MENU button to add a restaurant!" />
    </LinearLayout>
</FrameLayout>
```

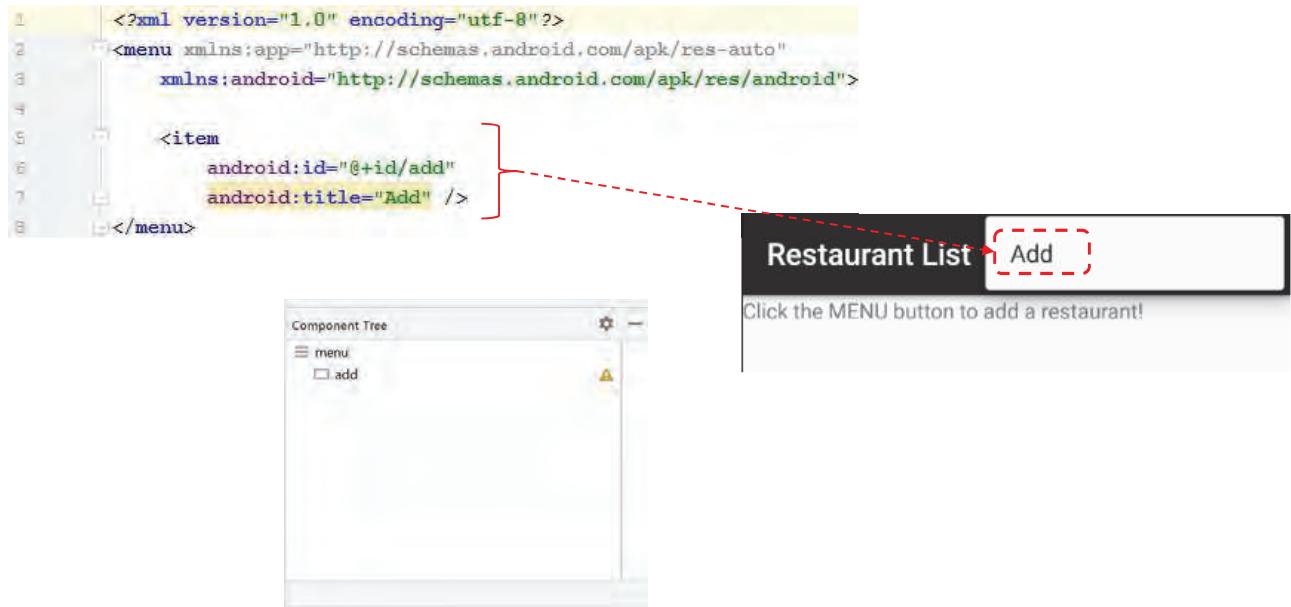
Saved in SQLite  
database



# View

## Add Menu Option

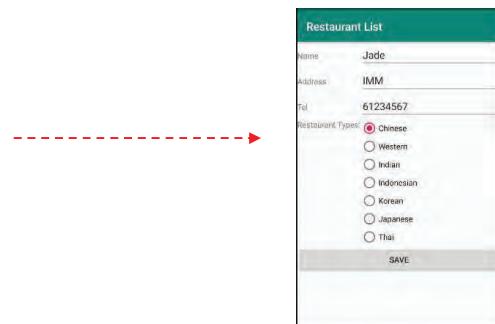
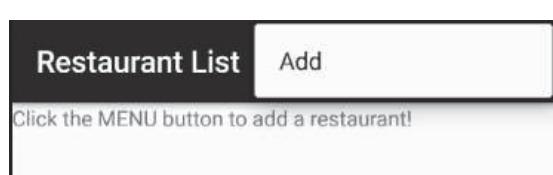
- The *option.xml* layout file under **res/menu** folder will be modified to show “Add” MENU option



# Controller – Explicit Intent & Option Menu

## Controller

- The Controller tasks are split and handled by individual activities (*RestaurantList* and *DetailForm*)
- *RestaurantList* takes care of displaying each restaurant record in the “restaurants\_table” in the *ListView view*. When the ‘Add’ MENU item is selected, it will then activate the *DetailForm Activity* to load the ‘Details’ form UI View to **add a new restaurant record**.



# Controller

- *RestaurantList* also takes care of *ListView* and *Cursor* Model update for ‘List’ UI View through *CursorAdapter*. Through the CursorAdapter (RestaurantAdapter), any insert, update or delete of records in the “restaurants\_table” will update the *ListView* view.
- *DetailForm Activity* handles the ‘Details’ form UI View and interaction (read, insert, update and delete) with the “restaurants\_table” Model through *RestaurantHelper* .

# Data Updating

## Activity

- In the second part of Practical 4 exercise, an extra feature is added to allow user to update an existing restaurant data and save the changes.

## Activity

- Let's check what do we need to modify from the previous exercise to allow a restaurant record to be updated?
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do anything new?

# Activity

Model - YES

Two new methods need to be added to the *RestaurantHelper*

```
/* Update a particular record in restaurants_table with id provided */
public void update(String id, String restaurantName, String restaurantAddress,
                    String restaurantTel, String restaurantType) {
    ContentValues cv = new ContentValues();
    String[] args = {id};
    cv.put("restaurantName", restaurantName);
    cv.put("restaurantAddress", restaurantAddress);
    cv.put("restaurantTel", restaurantTel);
    cv.put("restaurantType", restaurantType);

    getWritableDatabase().update("restaurants_table", cv, "_ID = ?", args);
}

public Cursor getById(String id) {
    String[] args = {id};

    return (getReadableDatabase().rawQuery(
        "SELECT _id, restaurantName, restaurantAddress, restaurantTel, " +
        "restaurantType FROM restaurants_table WHERE _ID = ?", args));
}
```

# Activity

View - NO

Controller - YES

- ✓ **onListClick** - The “OnItemClickListener” listener of the **RestaurantList** is to detect the item selection of an item in the ‘List’ UI View. It gets the **record “ID”** of the selected item, activate the *DetailForm* and pass the **“ID”** to the *DetailForm*’s Controller.

```
private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        model.moveToPosition(position);
        String recordID = helper.getID(model);
        Intent intent;
        intent = new Intent(RestaurantList.this, DetailForm.class);
        intent.putExtra("ID", recordID);
        startActivity(intent);
    }
};
```

# Model – RestaurantHelper



## Model

### RestaurantHelper Model

- There is no change in to the “restaurants\_table” model. The data attributes (fields) will remain unchanged
- Due to the extra features in **retrieving and updating existing record based on “ID”**, the *RestaurantHelper* will be **added with three new methods** for the purpose

# Model

## RestaurantHelper Model

- Method to retrieve record from the “restaurants\_table” model by record “\_id” field

```
/* Read a particular record from restaurants_table with the id provided */
public Cursor getById(String id) {
    String[] args = {id};

    return (getReadableDatabase().rawQuery(
        "SELECT _id, restaurantName, restaurantAddress, restaurantTel, " +
        "restaurantType FROM restaurants_table WHERE _ID = ?", args));
}
```

# Model

## RestaurantHelper Model

- Method to update existing record to the “restaurants\_table” model by record “\_id” field

```
public void update(String id, String restaurantName, String restaurantAddress,
                   String restaurantTel, String restaurantType) {
    ContentValues cv = new ContentValues();
    String[] args = {id};
    cv.put("restaurantName", restaurantName);
    cv.put("restaurantAddress", restaurantAddress);
    cv.put("restaurantTel", restaurantTel);
    cv.put("restaurantType", restaurantType);

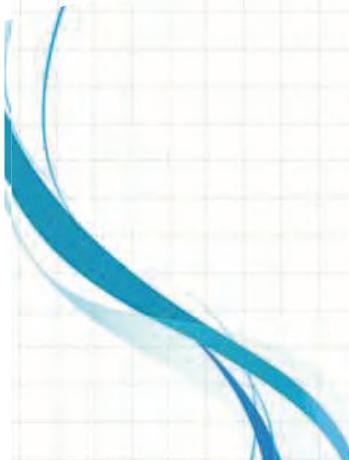
    getWritableDatabase().update("restaurants_table", cv, "_ID=?", args);
}
```

# Model

## RestaurantHelper Model

- Method to get the value of the “\_id” field of the record read (Cursor) from the “restaurants\_table” model. Note that the field index of the table model declaration starts from 0. Hence “`_id`”, has index 0, “`restaurantName`” has 1, “`restaurantAddress`” has 2, “`restaurantTel`” has 3 and “`restaurantType`” has 4.

```
public Cursor getById(String id) {  
    String[] args = {id};  
  
    return (getReadableDatabase() .rawQuery(  
        "SELECT _id, restaurantName, restaurantAddress, restaurantTel, restaurantType " +  
        "FROM restaurants_table WHERE _ID=?", args));  
}  
  
public String getID(Cursor c) {  
    return (c.getString(0));  
}
```



# Controller – 'List' Item Select & Passing 'ID'

## Controller Of RestaurantList

- Detect 'List' item click through OnItemClickListener and make an Explicit Intent call to *DetailForm*
- Pass the record ID over to the 'Details' form using Intent **putExtra** method.
- More about passing data between activities:
- <https://zocada.com/using-intents-extras-pass-data-activities-android-beginners-guide/>

```
private AdapterView.OnItemClickListener onListClick = new
    AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            model.moveToPosition(position);
            String recordID = helper.getID(model);
            Intent intent;
            intent = new Intent(RestaurantList.this, DetailForm.class);
            intent.putExtra("ID", recordID);
            startActivity(intent);
        }
    };
}
```

# Controller

- *DetailForm* Activity uses the `getIntent.getStringExtra` to retrieve the ‘ID’ string passed over by *RestaurantList* activity and save to local variable `restaurantID`

```
restaurantID = getIntent().getStringExtra("ID");
```

# Controller

- How does *DetailForm* able to differentiate whether the *Explicit Intent* calls from *RestaurantList* (Call 1 - ‘Add’ MENU item is selected;
- Call 2 - ‘List’ Item is selected) is **to add a new restaurant record or to update an existing restaurant record** when “Save” button is pressed?



# Controller Of DetailForm

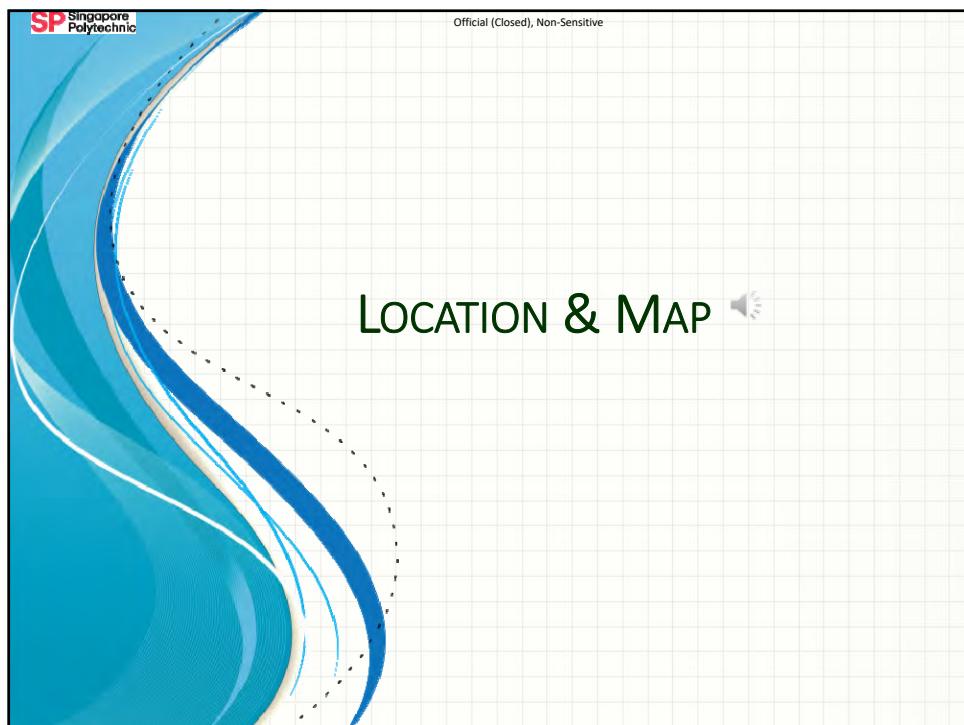
- The solution is to check the `restaurantID` value
- For adding new record, the Explicit Intent Call will not pass an 'ID' to 'Details' form i.e. `restaurantID equals to 'null'` (`helper.insert` method)
- If `restaurantID` has an 'ID' from the `RestaurantList` activity then update the record based on the 'ID' (`helper.update` method)

```
restaurantID = getIntent().getStringExtra("ID");

if (restaurantID == null) {
    helper.insert(nameStr, addrStr, telStr, restType);
} else {
    helper.update(restaurantID, nameStr, addrStr, telStr, restType);
}
```



END



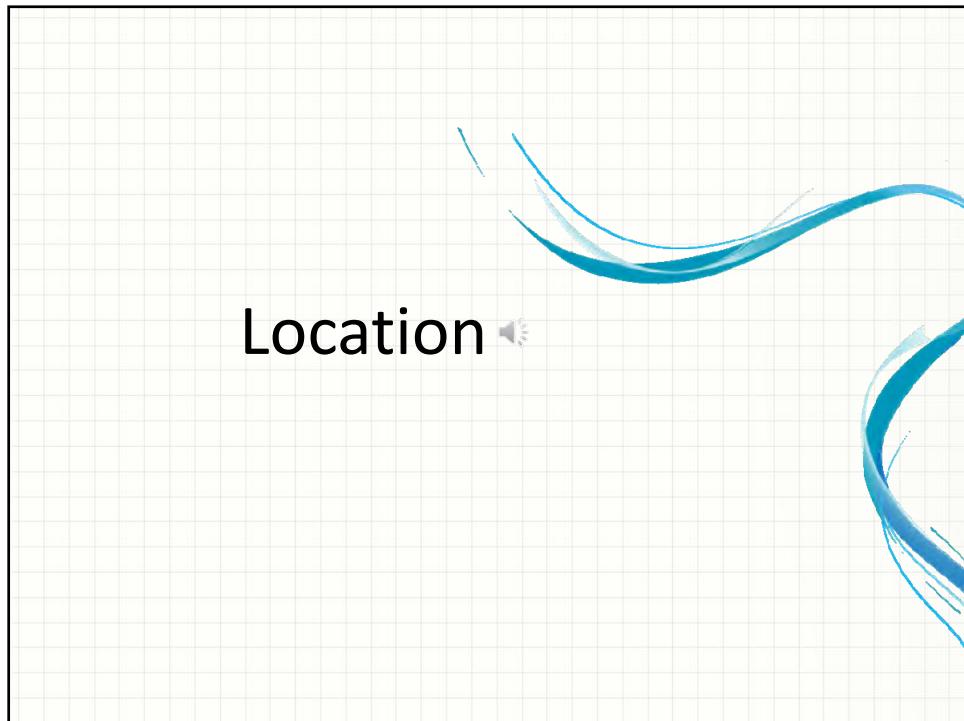
## Today's Overview

1

- Location

2

- Map



## Location

- In Practical 5a, an extra feature is added to the Restaurant List app to get the restaurant's location using the Android's **LocationManager** system service.
- The LocationManager system service, allow applications to obtain periodic updates of the device's geographical location, or to be notified when the device enters the proximity of a given geographical location.
- In Practical 5a, the LocationManager system service get the restaurant's location latitude and longitude.
- Together with the restaurant's name, address, telephone and restaurant type, the location latitude and longitude will be saved to the SQLite "restaurants\_table" as a record.

## Location



- Let's check what do we need to modify to get and save the restaurant's GPS coordinates (latitude & longitude).

**Model** - Any change in Data Model?

**View** - Do you need to modify any of the user interface view?

**Controller** - Do you need to tell the Controller to do any thing new?

## Location



**Model – YES**

Two new data fields are added to “restaurants\_table” to store the latitude and longitude of a restaurant's location, and also the corresponding handling methods are added to RestaurantHelper.java

## Location



### ❑View – YES

There will be 2 changes to the DetailForm view:

- ✓ For the *detail\_form.xml* layout, a *TextView* widget is added to display the restaurant's location information
- ✓ A new *details\_option.xml* layout for a MENU option View is created in *res/menu* folder. A MENU item "Get Location" is added to allow user to get the restaurant's GPS coordinates.



## Location



### ❑Controller – YES

The changes will be done at *DetailForm* Controller

- ✓ When MENU button is pressed, the *onCreateOptionsMenu(Menu)* is used to inflate the *details\_option.xml* layout displaying the "Get Location" menu item.
- ✓ *onOptionsItemSelected(MenuItem)* detects "Get Location" MENU option item to retrieve GPS location

## Location



### Controller – YES

The changes will be done at the *DetailForm* Controller

✓ **GPSTracker** object (which is a sub-class of *Service* with the **LocationListener** interface implemented) will be created upon the creation of the *DetailForm* Activity.

✓ **onDestroy()** callback methods is updated to stop the *GPSTracker* service and to close the SQLite database.

## Time-Out

- To get the location of the restaurant's location, what are the changes made?
  - Add 2 columns to the “restaurants\_table”.
  - Add a new menu option.
  - Add an menu option clicked event handler.
  - Add a new GPSTracker class.
  - Add a new form to display the restaurant's location.



View

DetailForm

- Two *TextView* widgets are added to the *detail\_form.xml* to display the GPS location information

```
<TableRow  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Location" />  
  
    <TextView  
        android:id="@+id/location"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content" />  
/</TableRow>
```

The screenshot shows a list of restaurants including KFC, Food Court 5, and 1234567. Below the list is a section titled 'Restaurant Type' with checkboxes for Chinese, Indian, Indonesian, Korean, Japanese, and Thai. The 'Wendy's' checkbox is checked. At the bottom of the screen, there is a 'Search' bar and a 'SAVE' button.

## View



### DetailForm - MENU Option

- A *details\_option.xml* layout is created for the MENU option in res/menu

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

    <item
        android:id="@+id/get_location"
        android:title="Get Location" />

</menu>
```



Model – RestaurantHelper

## Model



### RestaurantHelper

- “restaurants\_table” model is modified to store latitude and longitude of a restaurant

```
@Override  
public void onCreate(SQLiteDatabase db) {  
    // Will be called once when the database is not created  
    db.execSQL("CREATE TABLE restaurants_table (_id INTEGER PRIMARY KEY AUTOINCREMENT, "  
              + "name TEXT, address TEXT, telephone TEXT, restaurantType TEXT, "  
              + "lat REAL, lon REAL);");  
}
```

## Model



### RestaurantHelper

- For upgrading the SQLite database of an existing app, the **SCHEMA\_VERSION** is required to increase. When the existing SCHEMA\_VERSION (old version) **is smaller** than the new SCHEMA\_VERSION, the **onUpgrade()** method will be called on the first initialization of the RestaurantHelper object.
- **Note that this is not implemented in Practical 5a and 5b.**

# Model

## RestaurantHelper

- In Practical 4, in the RestaurantHelper.java, **SCHEMA VERSION** has the value of **1**.

```
public class RestaurantHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "restaurantlist.db";
    private static final int SCHEMA_VERSION = 1;
```

- For example, in Practical 5a, if we set the value of **SCHEMA\_VERSION** to **2**, then on the first initialization of RestaurantHelper, the **onUpgrade()** method below will be executed and the “**restaurants\_table**” will be upgraded with the latitude and longitude fields. By upgrading existing records of “**restaurants\_table**” in the database will not be deleted.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion < SCHEMA_VERSION) {
        db.execSQL("ALTER TABLE restaurants_table ADD COLUMN lat REAL");
        db.execSQL("ALTER TABLE restaurants_table ADD COLUMN lon REAL");
    }
}
```

# Model



## RestaurantHelper

- The two query methods, **getAll()** and **getById()**, are updated to return the latitude and longitude data from the restaurants table Model

```
/* Read all records from restaurants_table */
public Cursor getAll() {
    return (getReadableDatabase()).rawQuery(
        "SELECT _id, restaurantName, restaurantAddress, restaurantTel, " +
        "restaurantType, lat, lon FROM restaurants_table ORDER BY restaurantName", null);
}

/* Read a particular record from restaurants_table with id provided */
public Cursor getById(String id) {
    String[] args = {id};

    return (getReadableDatabase()).rawQuery(
        "SELECT _id, restaurantName, restaurantAddress, restaurantTel, " +
        "restaurantType, lat, lon FROM restaurants_table WHERE _ID=?", args);
}
```

## Model



### RestaurantHelper

- Method **insert()** is updated to include inserting the latitude and longitude values when a new record is inserted to the “**restaurants\_table**” model.

```
/* Write a record into restaurants_table */
public void insert(String restaurantName, String restaurantAddress, String restaurantTel,
                   String restaurantType, double lat, double lon) {
    ContentValues cv = new ContentValues();

    cv.put("restaurantName", restaurantName);
    cv.put("restaurantAddress", restaurantAddress);
    cv.put("restaurantTel", restaurantTel);
    cv.put("restaurantType", restaurantType);
    cv.put("lat", lat);
    cv.put("lon", lon);

    getWritableDatabase().insert("restaurants_table", "restaurantName", cv);
}
```

## Model



### RestaurantHelper

- Method **update()** is updated to include updating of the latitude and longitude values to an existing record in the “**restaurants\_table**” model.

```
/* Update a particular record in restaurants_table with id provided */
public void update(String id, String restaurantName, String restaurantAddress,
                   String restaurantTel, String restaurantType, double lat, double lon) {
    ContentValues cv = new ContentValues();
    String[] args = {id};
    cv.put("restaurantName", restaurantName);
    cv.put("restaurantAddress", restaurantAddress);
    cv.put("restaurantTel", restaurantTel);
    cv.put("restaurantType", restaurantType);
    cv.put("lat", lat);
    cv.put("lon", lon);

    getWritableDatabase().update("restaurants_table", cv, "_ID=?", args);
}
```

## Model



### RestaurantHelper

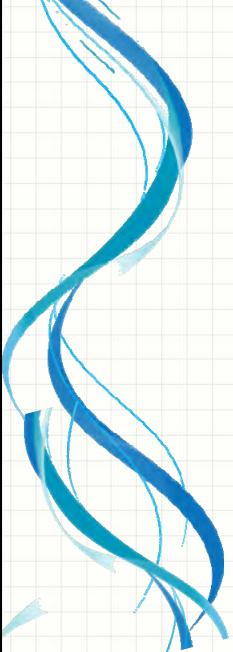
- Two new **getter** methods are added to return the latitude and longitude data from the **Cursor** model. Take note that the order of the index number is in accordance to the order of the attributes of the select query starting from **0**.

```
SELECT _id, name, address, telephone, restaurantType, lat, lon FROM restaurants_table
    0   1   2   3   4   5   6
    public double getLatitude(Cursor c) {
        return (c.getDouble(5));
    }

    public double getLongitude(Cursor c) {
        return (c.getDouble(6));
    }
```

## Time-Out

- What are the methods affected when 2 new columns are added to the “restaurants\_table”?
  - getAll()
  - getById()
  - insert()
  - update()
  - getLocation()
  - getLatitude()
  - getLongitude()



## Controller – DetailForm

### Controller

- GPS radio is normally not on. On instantiation of *DetailForm*, in the *OnCreate()* method it instantiate the *GPSTracker* which in turn instantiate *LocationManager* for getting location update through the *getSystemService()* method.

```
DetailForm { private GPSTracker gpsTracker;  
gpsTracker = new GPSTracker(DetailForm.this);  
  
GPSTracker { // Declaring a Location Manager  
protected LocationManager locationManager;  
locationManager = (LocationManager) mContext  
    .getSystemService(LOCATION_SERVICE);
```

## Controller



- Upon creation of the **GPSTracker** service, the **LocationManager** will first check on the **GPS** and **network** status before starting to fetch location data via the **requestLocationUpdates()** method.

```
GPSTracker {
    //Getting GPS status
    //This provider determines location using satellites.
    //Depending on conditions, this provider may take a while
    //to return a location fix. Requires the permission ACCESS_FINE_LOCATION.
    isGPSEnabled = locationManager
        .isProviderEnabled(LocationManager.GPS_PROVIDER);

    //Getting network status
    //This provider determines location based on availability of cell tower
    //and WiFi access points. Results are retrieved by means of a network lookup
    isNetworkEnabled = locationManager
        .isProviderEnabled(LocationManager.NETWORK_PROVIDER);
```

## Controller

- If user has enabled the **network provider** (mobile data or WiFi) in the phone Settings menu, the **LocationManager** will fetch the location data based on the availability of **cell towers** and **WiFi** access points.

```
if (isNetworkEnabled) {
    locationManager.requestLocationUpdates(
        LocationManager.NETWORK_PROVIDER,
        MIN_TIME_BW_UPDATES,
        MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
    Log.d("Network", "Network");
    if (locationManager != null) {
        location = locationManager
            .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
        if (location != null) {
            latitude = location.getLatitude();
            longitude = location.getLongitude();
        }
    }
}
```

## Controller

- On the phone, under the settings menu, if the user has enabled the **GPS provider**, the **LocationManager** will fetch the location data using **satellites**. Depending on conditions, this provider may take a while to return a location fix.

```
if (isGPSEnabled) {
    if (location == null) {
        locationManager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            MIN_TIME_BW_UPDATES,
            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
        Log.d("GPS Enabled", "GPS Enabled");
        if (locationManager != null) {
            location = locationManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER);
            if (location != null) {
                latitude = location.getLatitude();
                longitude = location.getLongitude();
            }
        }
    }
}
```

## Controller

- On the DetailForm, when user taps on the “Get Location” MENU item, the GPSTracker **getLatitude()** and **getLongitude()** methods will be called that return the **latitude** and **longitude** data respectively.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.get_location) {
        if (gpsTracker.canGetLocation()) {
            latitude = gpsTracker.getLatitude();
            longitude = gpsTracker.getLongitude();
            location.setText(String.valueOf(latitude) + ", " + String.valueOf(longitude));
            // \n is for new line
            Toast.makeText(getApplicationContext(), "Your Location is - \nLat: " + latitude
                + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
        }
        return (true);
    }
    return super.onOptionsItemSelected(item);
}
```

## Controller

- It is possible that user will exit the *DetailForm* activity (i.e. when BACK button or “Save” is pressed) while waiting on a GPS fix. The DetailForm Controller will need to stop requesting GPS updates by stopping the *GPSTracker* service (calling *stopUsingGPS()* of *GPSTracker*) in the *onDestroy()* event of DetailForm.

```
//  
// * Stop using GPS listener  
// * Calling this function will stop using GPS in your app  
// */  
  
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    helper.close();  
    gpsTracker.stopUsingGPS();  
}  
  
public void stopUsingGPS() {  
    if (locationManager != null) {  
        locationManager.removeUpdates(GPSTracker.this);  
    }  
}
```

## Controller

### AndroidManifest.xml

- In order to allow *LocationManager* to have access to the **GPS information**, permissions must be set in the *AndroidManifest.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.sp.restaurantlist">  
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
    <uses-permission android:name="android.permission.INTERNET" />  
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

## Controller

### AndroidManifest.xml

- **android.permission.ACCESS\_COARSE\_LOCATION** – Allows the API to use WiFi or mobile cell data (or both) to determine the device's location. The API returns the location with an accuracy approximately equivalent to a city block.
- **android.permission.ACCESS\_FINE\_LOCATION** – Allows the API to determine as precise a location as possible from the available location providers, including the Global Positioning System (GPS) as well as WiFi and mobile cell data.
- **android.permission.INTERNET** – allows access to the internet.

## Controller

### Request Runtime Permissions – GPSTracker

- Since the app needs the location permission, you must check to see if it has the necessary permissions at runtime, and request the permission if it does not have it. If not displays a dialog to ask the user for the permission.

```
private void getLocation() {  
    this.canGetLocation = false;  
    int permissionState1 = ActivityCompat.checkSelfPermission(mContext,  
        Manifest.permission.ACCESS_FINE_LOCATION);  
    int permissionState2 = ActivityCompat.checkSelfPermission(mContext,  
        Manifest.permission.ACCESS_COARSE_LOCATION);  
    if (permissionState1 == PackageManager.PERMISSION_GRANTED &&  
        permissionState2 == PackageManager.PERMISSION_GRANTED) {  
        // Permission granted, get GPS location  
        locationManager = (LocationManager) mContext.getSystemService(Context.LOCATION_SERVICE);  
        location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);  
        if (location != null) {  
            latitude = location.getLatitude();  
            longitude = location.getLongitude();  
        }  
        this.canGetLocation = true;  
    } else {  
        // Permission not granted, prompt user to enable location permission  
        ActivityCompat.requestPermissions((Activity)mContext, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},  
            GPS_PERMISSION_REQUEST_CODE);  
    }  
}
```

# Controller

## Get Location Longitude and Latitude— GPSTracker

- When location permission is granted.

```
private static final int GPS_PERMISSION_REQUEST_CODE = 1001;

private void getLocation() {
    this.canGetLocation = false;
    int permissionState1 = ActivityCompat.checkSelfPermission(mContext,
        Manifest.permission.ACCESS_FINE_LOCATION);
    int permissionState2 = ActivityCompat.checkSelfPermission(mContext,
        Manifest.permission.ACCESS_COARSE_LOCATION);
    if (permissionState1 == PackageManager.PERMISSION_GRANTED &&
        permissionState2 == PackageManager.PERMISSION_GRANTED) {
        // Permission granted, get GPS location
        locationManager = (LocationManager) mContext.getSystemService(Context.LOCATION_SERVICE);
        location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        if (location != null) {
            latitude = location.getLatitude();
            longitude = location.getLongitude();
        }
        this.canGetLocation = true;
    } else {
        // Permission not granted, prompt user to enable location permission
        ActivityCompat.requestPermissions((Activity)mContext, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
            GPS_PERMISSION_REQUEST_CODE);
    }
}
```

# Controller

## Request Runtime Permissions – GPSTracker

- Dialog box to prompt user to enable location permission.

```
private static final int GPS_PERMISSION_REQUEST_CODE = 1001;

// Permission not granted, prompt user to enable location permission
ActivityCompat.requestPermissions((Activity)mContext, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
    GPS_PERMISSION_REQUEST_CODE);
```

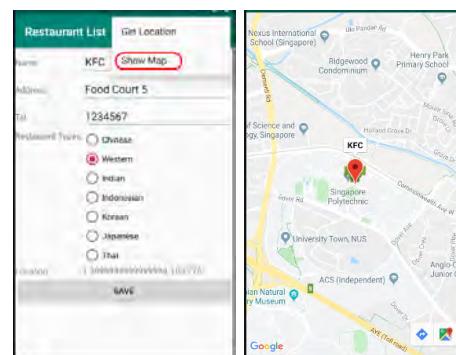
## Time-Out

- To get the location using the GPS satellite, WiFi and mobile data what permissions need to add to the AndroidManifest.xml?

MAP 

## Map

- In Practical 5b, using the latitude and longitude captured and saved in Practical 5a for each restaurant, we will show each restaurant location and the user's current location on the Google map.



## Map

- Let's check what do we need to modify from the previous exercise to show restaurant and user current location using Google map.
  - Model** - Any change in Data Model?
  - View** - Do you need to modify any of the user interface view?
  - Controller** - Do you need to tell the Controller to do any thing new?

## Map



### Model – NO

Since there are no changes to the data Model and the handling methods, there will be no change for RestaurantHelper.java

## Map



### View – YES

- A “Show Map” MENU item is added to `details_option.xml` layout, that will be displayed in the `DetailForm` menu.
- A new fragment activity, `RestaurantMap` is created to display the Google map with its layout, `activity_restaurant_map.xml` created in the `res/layout` folder.

## Map

### Controller – YES

There are two changes at the Controller

#### 1. DetailForm Controller

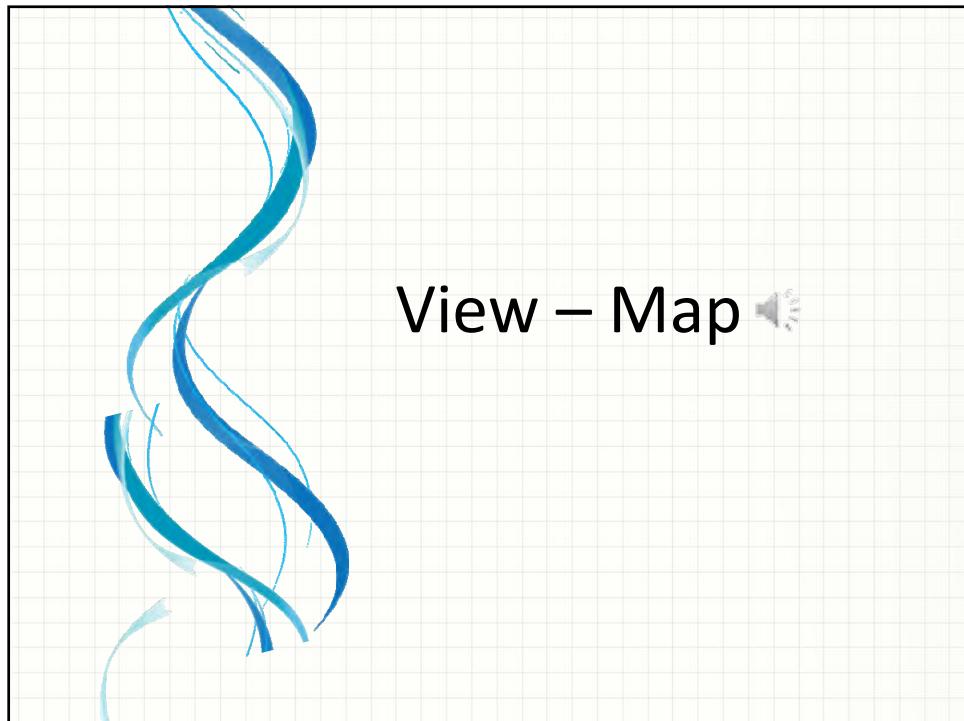
Update `onOptionsItemSelected(MenuItem)` to detect the “Show Map” MENU option item, which when selected, will do an Explicit Intent call to the `RestaurantMap` FragmentActivity.

#### 2. RestaurantMap Controller

To handle the display of a restaurant and user’s locations on the Google map.

## Time-Out

- Identify the controller method that is responsible to display the Google map.
- Which activity serve to display the restaurant’s location and current user’s location on the Google map.



## View

- An extra “ Show Map” item is added to the *details\_option.xml* layout which when selected display a map.

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

    <item
        android:id="@+id/get_location"
        android:title="Get Location" />
    <item
        android:id="@+id/show_map"
        android:title="Show Map" />
</menu>
```



## View

- For viewing map, an *activity\_restaurant\_map.xml* layout is created in **res/layout** folder
  - **Fragment** is a part of an **activity**, which contributes its own **UI** to that activity. Fragment can be thought like a **sub activity**, whereas the complete screen with which user interacts is called an activity. An activity can contain multiple fragments. Fragments are part of an activity.
  - Here the activity is **RestaurantMap** and the fragment is **SupportMapFragment**.

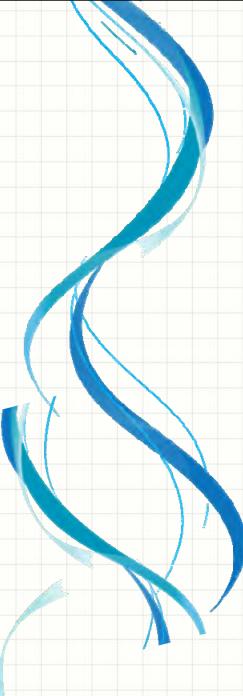
## View

- **SupportMapFragment** – This is a **Map** component in an app. This fragment is the simplest way to place a map in an application. It is a wrapper around a view of a map to automatically handle the necessary life cycle needs.
- Being a fragment, this component can be added to an activity's layout file simply with the XML below.

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/restaurant_map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RestaurantMap" />
```

## Time-Out

- Which component of the RestaurantMap activity is responsible to display the Google map?
- What change do you need to make to add this component to the RestaurantMap activity?



Controller –  
DetailForm &  
RestaurantMap 

# Controller

## DetailForm



- Update `onOptionsItemSelected()` method to capture “Show Map” MENU item pressed.
- Get the user’s current location.

```
    } else if (item.getItemId() == R.id.show_map) {  
        //Get my current location  
        myLatitude = gpsTracker.getLatitude();  
        myLongitude = gpsTracker.getLongitude();  
  
        Intent intent = new Intent(this, RestaurantMap.class);  
        //Restaurant location  
        intent.putExtra("LATITUDE", latitude);  
        intent.putExtra("LONGITUDE", longitude);  
        //User's location  
        intent.putExtra("MYLATITUDE", myLatitude);  
        intent.putExtra("MYLONGITUDE", myLongitude);  
        intent.putExtra("NAME", restaurantName.getText().toString());  
        startActivity(intent);  
        return (true);  
    }
```

# Controller

## DetailForm



- Create an **explicit intent call** to `RestaurantMap` activity.
- Pass the restaurant location and user’s location to `RestaurantMap` activity using the `putExtra()` method.

```
    } else if (item.getItemId() == R.id.show_map) {  
        //Get my current location  
        myLatitude = gpsTracker.getLatitude();  
        myLongitude = gpsTracker.getLongitude();  
  
        Intent intent = new Intent(this, RestaurantMap.class);  
        //Restaurant location  
        intent.putExtra("LATITUDE", latitude);  
        intent.putExtra("LONGITUDE", longitude);  
        //User's location  
        intent.putExtra("MYLATITUDE", myLatitude);  
        intent.putExtra("MYLONGITUDE", myLongitude);  
        intent.putExtra("NAME", restaurantName.getText().toString());  
        startActivity(intent);  
        return (true);  
    }
```

## Controller

### RestaurantMap



- The **RestaurantMap** Controller is responsible for setting up the UI View and plotting the **restaurant** and **user's** location on the Google map with markers.



## Controller

### RestaurantMap



- Get the restaurant and user's locations (latitude and longitude) transfer from DetailForm.
- Get ready the Map component (SupportMapFragment).

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

    lat = getIntent().getDoubleExtra("LATITUDE", 0);
    lon = getIntent().getDoubleExtra("LONGITUDE", 0);
    restaurantName = getIntent().getStringExtra("NAME");
    myLat = getIntent().getDoubleExtra("MYLATITUDE", 0);
    myLon = getIntent().getDoubleExtra("MYLONGITUDE", 0);

    // Obtain the SupportMapFragment and get notified when the map is ready to be used
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
        .findFragmentById(R.id.restaurant_map);
    mapFragment.getMapAsync(this);
}
```

## Controller

RestaurantMap 

- Using the restaurant latitude and longitude received from the DetailForm, create a restaurant **LatLng** object.
- Using the user's latitude and longitude received from the DetailForm, create a user **LatLng** object.

```
@Override  
public void onMapReady(GoogleMap googleMap) {  
    mMap = googleMap;  
  
    RESTAURANT = new LatLng(lat, lon);  
    ME = new LatLng(myLat, myLon);  
  
    Marker restaurant = mMap.addMarker(new MarkerOptions().position(RESTAURANT).title(restaurantName));  
    Marker me = mMap.addMarker(new MarkerOptions().position(ME).title("ME")  
        .snippet("My location")  
        .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_me)));  
  
    // Move the camera instantly to restaurant with a zoom of 15.  
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(RESTAURANT, 15));  
}
```

## Controller

RestaurantMap 

- Plot and place a marker and a title of the restaurant location on the Google map.

```
Marker restaurant = mMap.addMarker(new MarkerOptions()  
    .position(RESTAURANT)  
    .title(restaurantName));
```



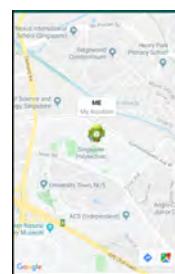
## Controller



### RestaurantMap

- Plot and place a marker and a title of the user's location on the Google map.

```
Marker me = mMap.addMarker(new MarkerOptions().position(ME).title("ME")
    .snippet("My location")
    .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_me)));
```



## Controller



### RestaurantMap

- Zoom in to the location with a specified zoom factor.

```
// Move the camera instantly to restaurant with a zoom of 15.
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(RESTAURANT, 15));
```

## Controller

### RestaurantMap



```
public class RestaurantMap extends FragmentActivity implements OnMapReadyCallback {  
    private GoogleMap mMap;  
    private double lat;  
    private double lon;  
    private String restaurantName;  
    private double myLat;  
    private double myLon;  
    private LatLng RESTAURANT;  
    private LatLng ME;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_restaurant_map);  
  
        lat = getIntent().getDoubleExtra("LATITUDE", 0);  
        lon = getIntent().getDoubleExtra("LONGITUDE", 0);  
        restaurantName = getIntent().getStringExtra("NAME");  
        myLat = getIntent().getDoubleExtra("MYLATITUDE", 0);  
        myLon = getIntent().getDoubleExtra("MYLONGITUDE", 0);  
  
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.  
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()  
            .findFragmentById(R.id.restaurant_map);  
        mapFragment.getMapAsync(this);  
    }  
  
    @Override  
    public void onMapReady(GoogleMap googleMap) {  
        mMap = googleMap;  
  
        RESTAURANT = new LatLng(lat, lon);  
        ME = new LatLng(myLat, myLon);  
  
        Marker restaurant = mMap.addMarker(new MarkerOptions().position(RESTAURANT).title(restaurantName));  
        Marker me = mMap.addMarker(new MarkerOptions().position(ME).title("ME")  
            .snippet("My location")  
            .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_me)));  
  
        // Move the camera instantly to restaurant with a zoom of 15.  
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(RESTAURANT, 15));  
    }  
}
```

## Controller



### AndroidManifest.xml

- To use Google Map in the Restaurant List app,
  - A valid **Google Map API** key needs to be included
  - The key is unique to individual computer

```
<meta-data  
    android:name="com.google.android.geo.API_KEY"  
    android:value="AIzaSyBD5a5zQ6Q3N14jIqvLDqoBtSvV1coICMg" />
```

## Controller

### AndroidManifest.xml

- For any new *Activity* (*RestaurantMap* Activity) created it has to be registered in the manifest file. Otherwise, an error will occur during run time

```
<activity
    android:name=".RestaurantMap"
    android:label="Map"></activity>

<activity android:name=".About" />
<activity android:name=".DetailForm" />
<activity android:name=".RestaurantList">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

## Time-Out

- When the “Show Map” menu option is selected, what data are passed from the DetailForm activity to the RestaurantMap activity?
- What is the purpose of the `putExtra()` method?
- What is the purpose of the `getIntent()` method?
- What are the purposes of the `getDoubleExtra()` the `getStringExtra()` methods?
- What is the purpose of the `getIntExtra()` method?

## Lab – Getting Started

### Introduction

Android Studio is the official IDE (Integrated Development Environment) for developing Android Apps by Google. It is based on JetBrains IntelliJ IDEA software and has lots of amazing features which helps developer in creating Android App.

Android Studio is available for free download on Windows, Mac OS X and Linux.

### A. Android Studio Installation

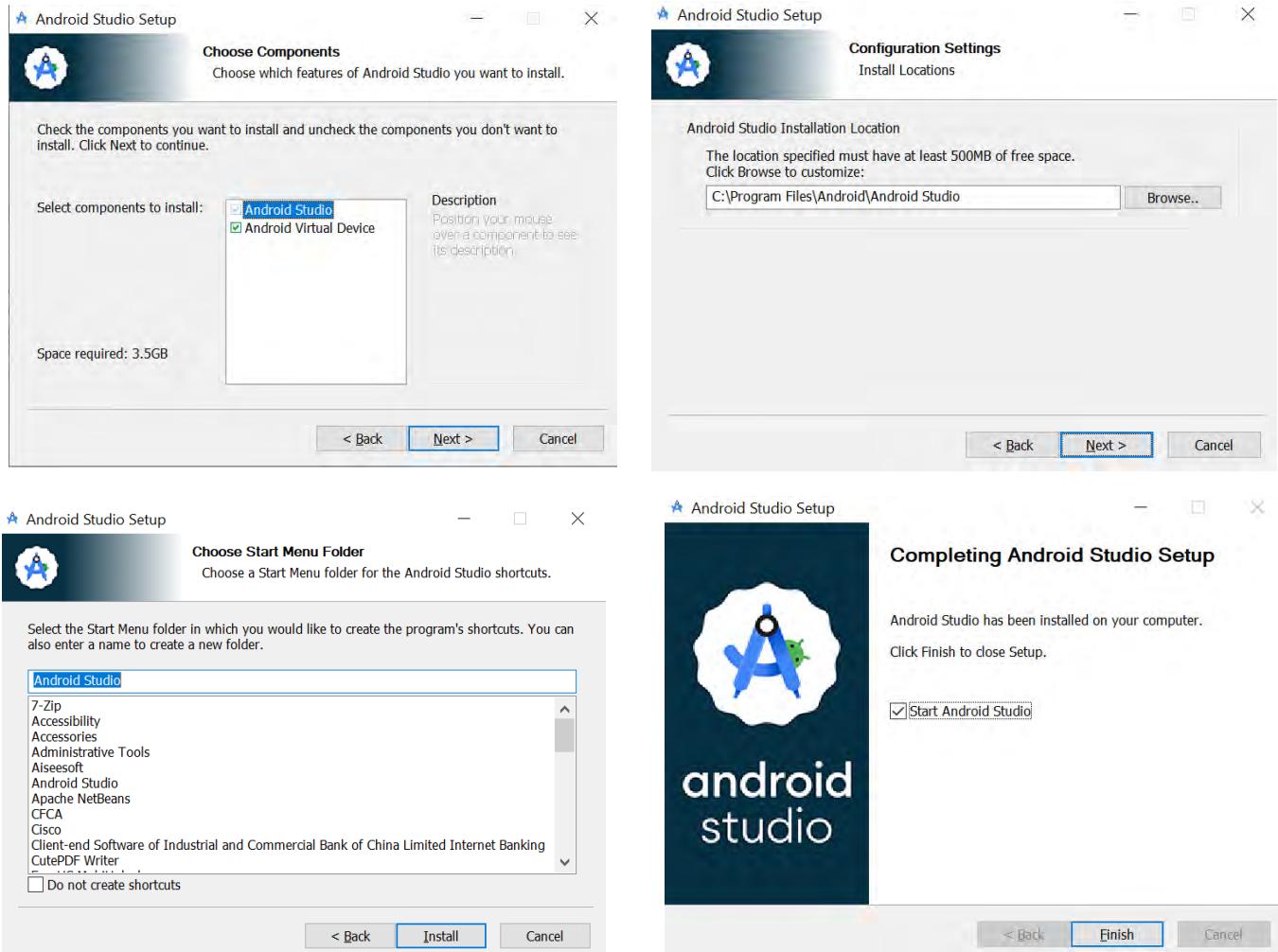
1. Download the Android Studio **Giraffe | 2022.3.1 Patch 4** installer from:

<https://developer.android.com/studio/archive>

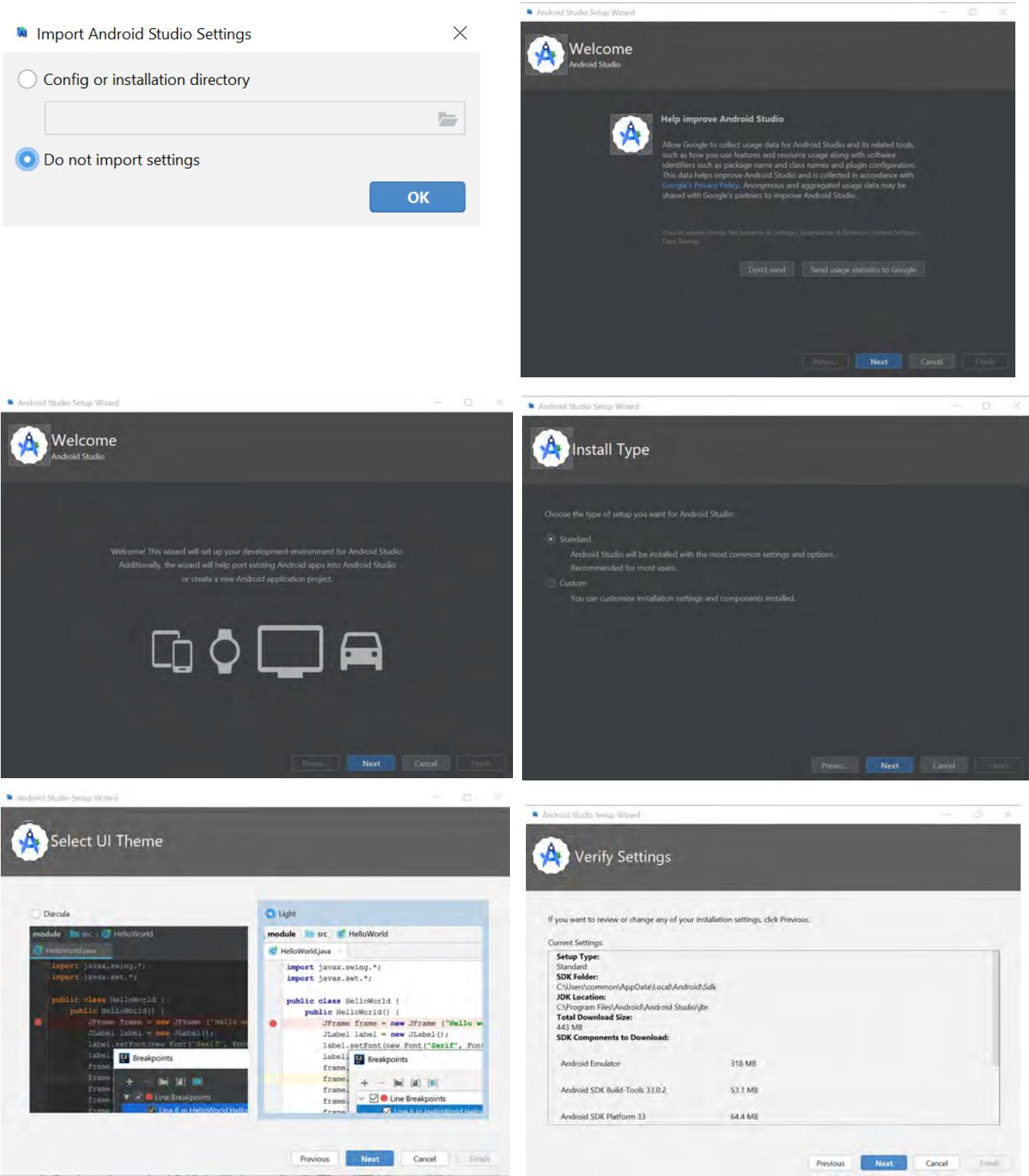
2. Double click on the installer downloaded to start installing the IDE using all default settings.

[Note: Must have Internet connection as some files will be downloaded from remote during installation]

[Note: The GUI and sequence of the installation process may differ depending on the Android version downloaded]

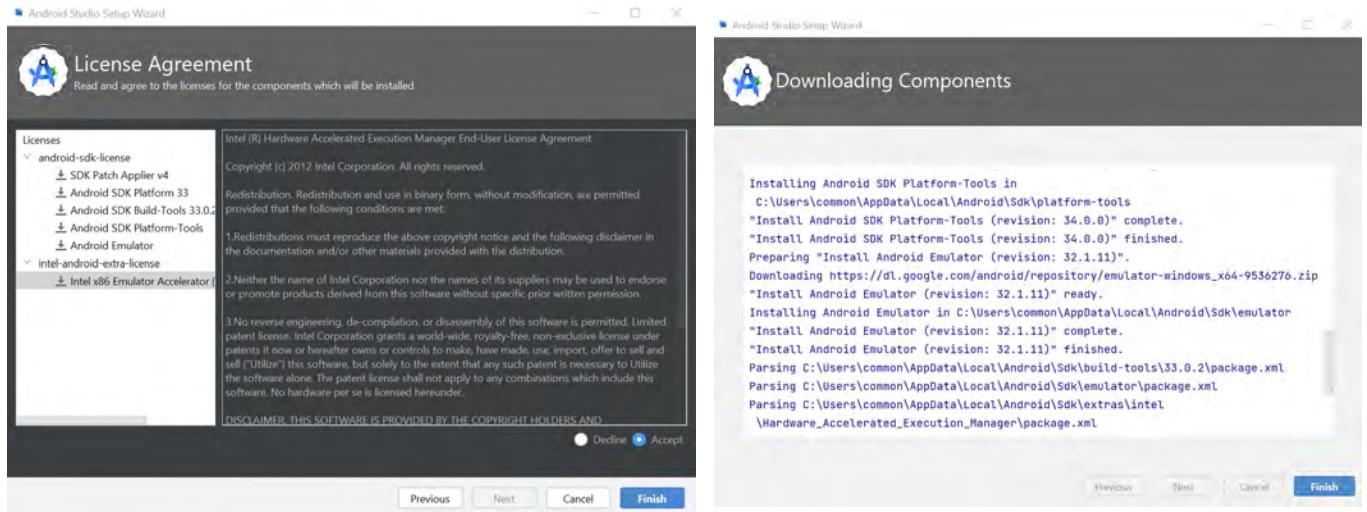


**Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering**



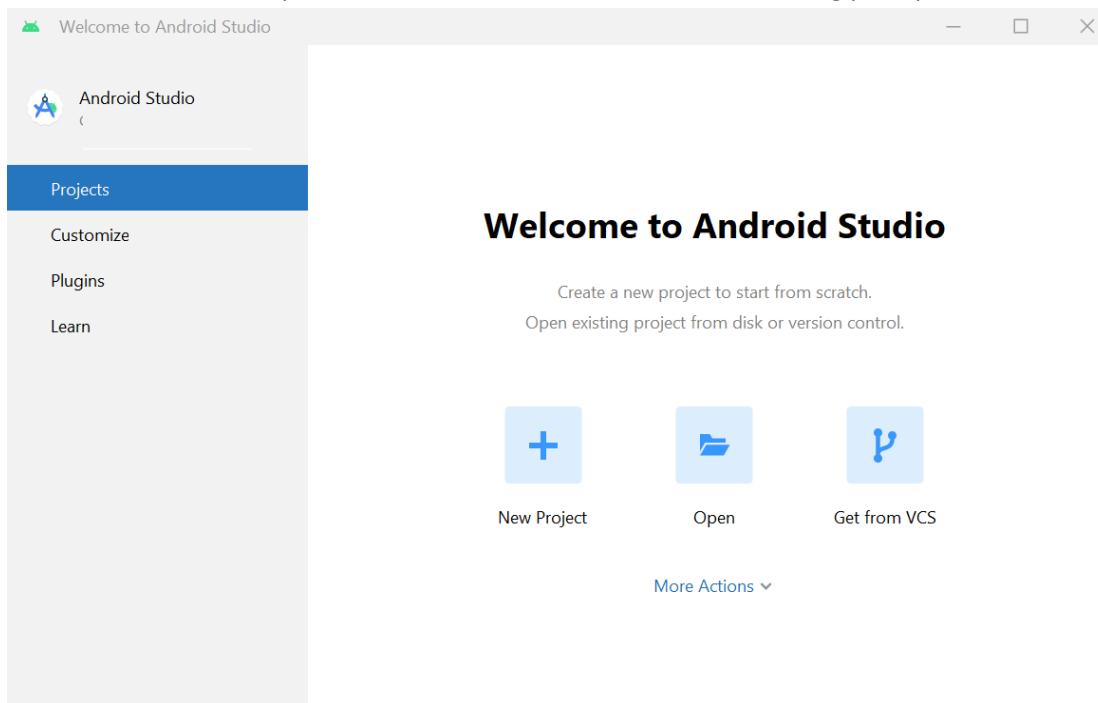
S

**Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering**



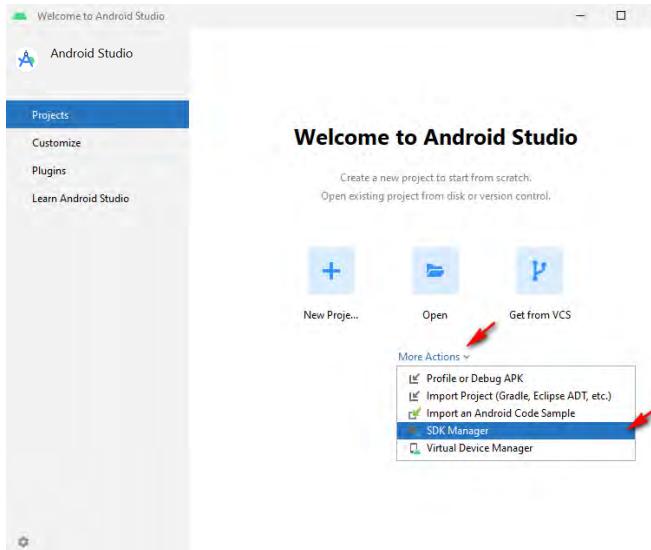
In the “**Downloading Components**” dialog prompt, be patient and let the installer download the necessary software components. This will take quite some time. Click “Finish” when complete.

3. Android Studio starts up with the “Welcome to Android Studio” dialog prompt.

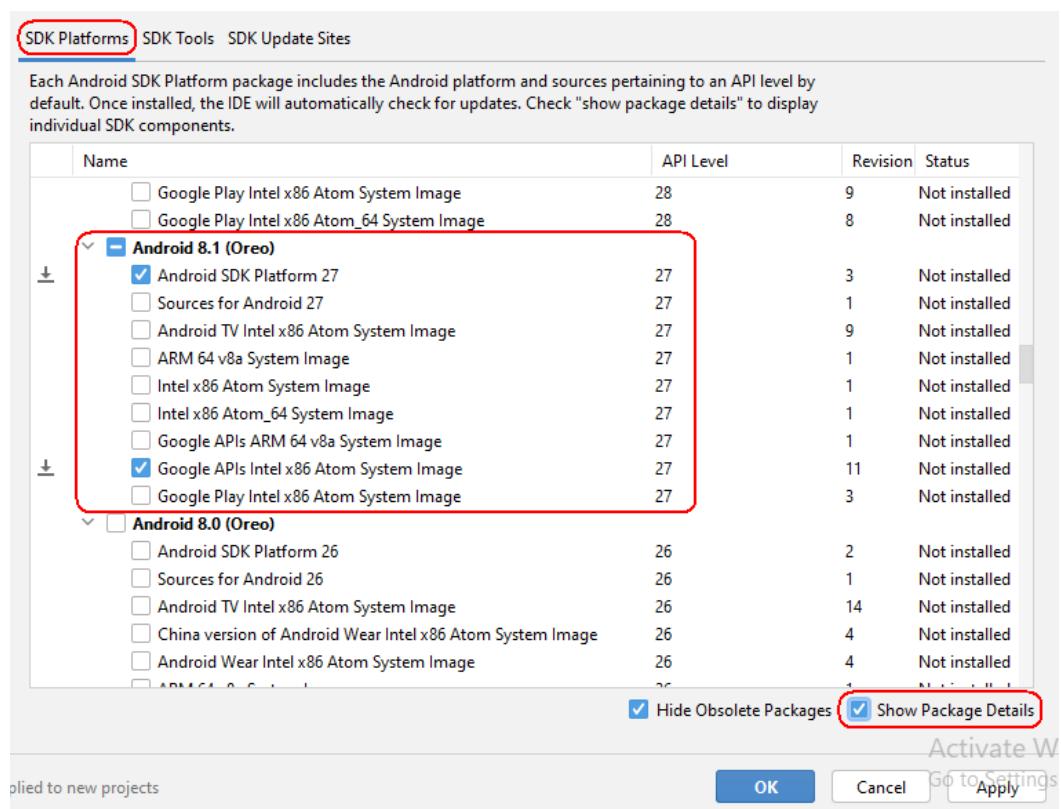


## B. Download Extra API

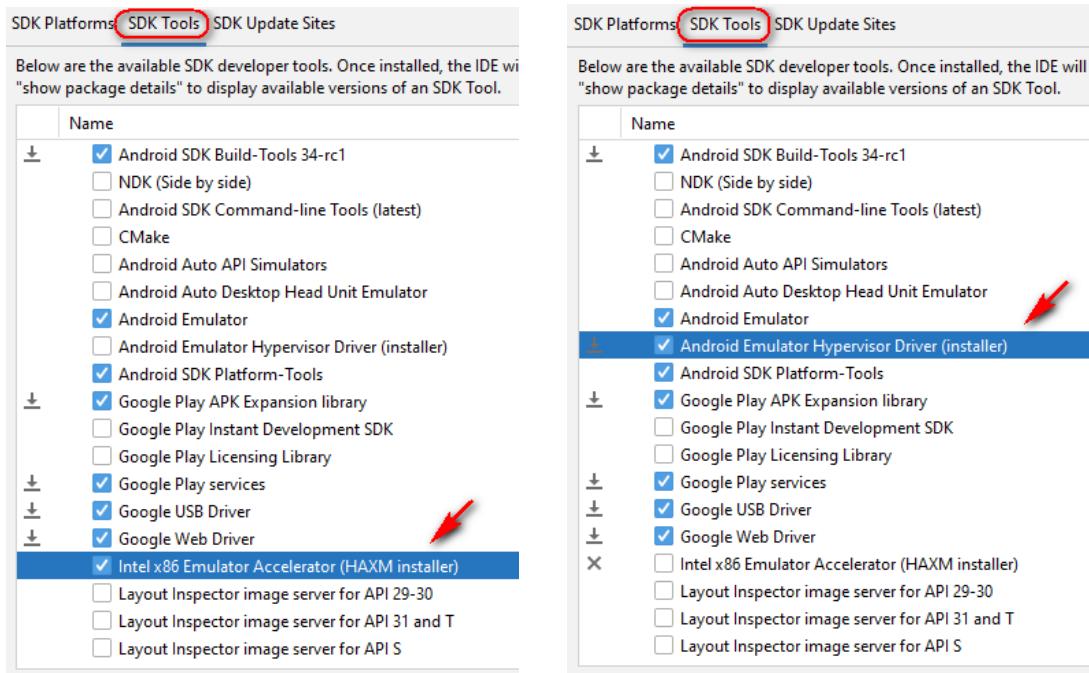
- Click the “More Actions” dropdown list and select “SDK Manager”.



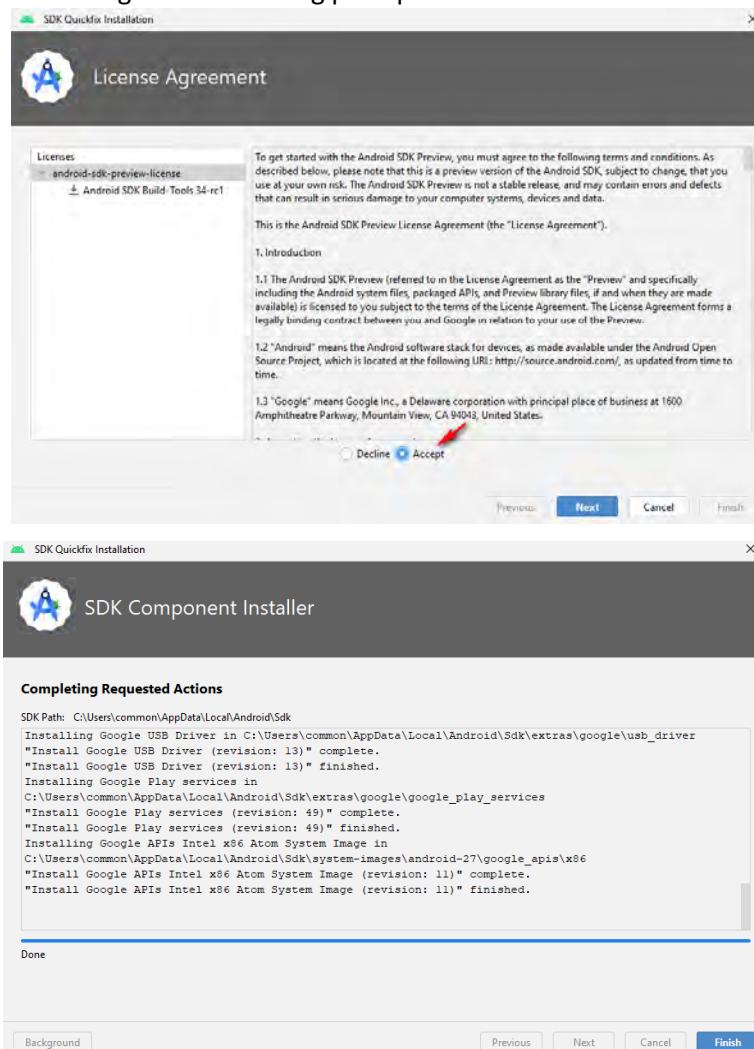
- Under **SDK Platform tab**, check **Show Package Details** and select the following components for **Android 8.1 (Oreo)**. Do not click “OK”.



3. At the “SDK Tools” tab, for “**Intel Processor**” system, select the components on the left diagram; for “**AMD Processor**” system, select the components on the right diagram to download.

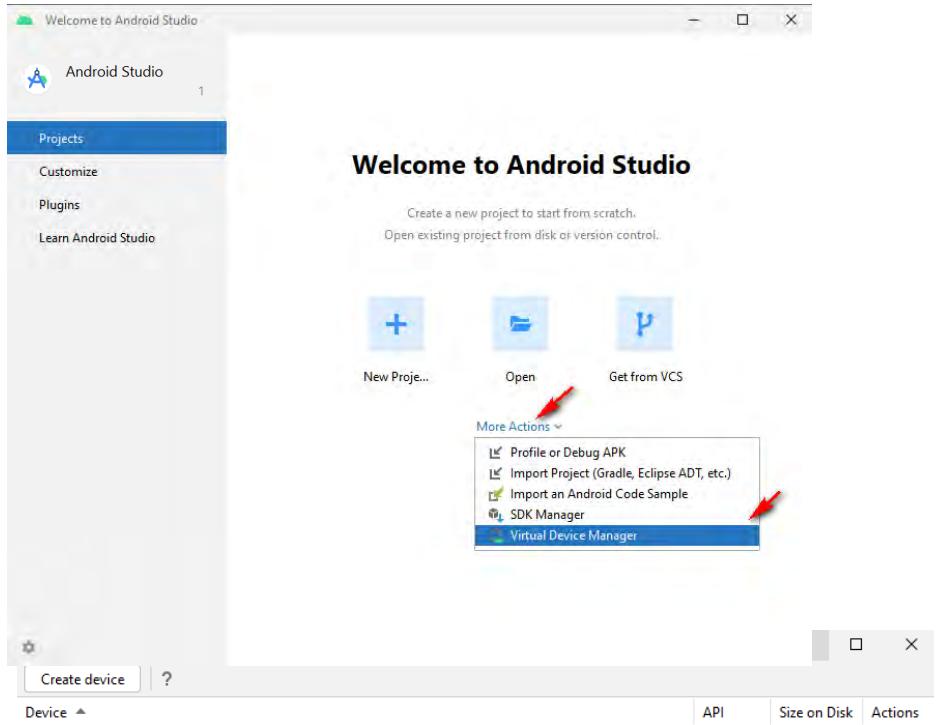


4. Click “OK”. For the “**Confirm Change**” dialog box, click “OK”. Accept the licenses for all the components in the “License Agreement” dialog prompt and click “Next” to start downloading.

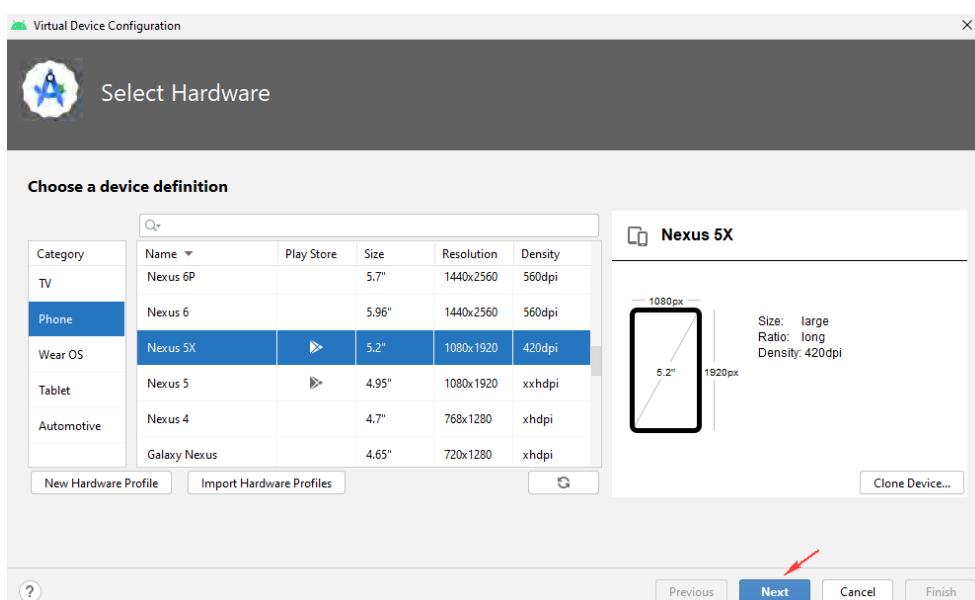


## C. Create new Android Virtual Device

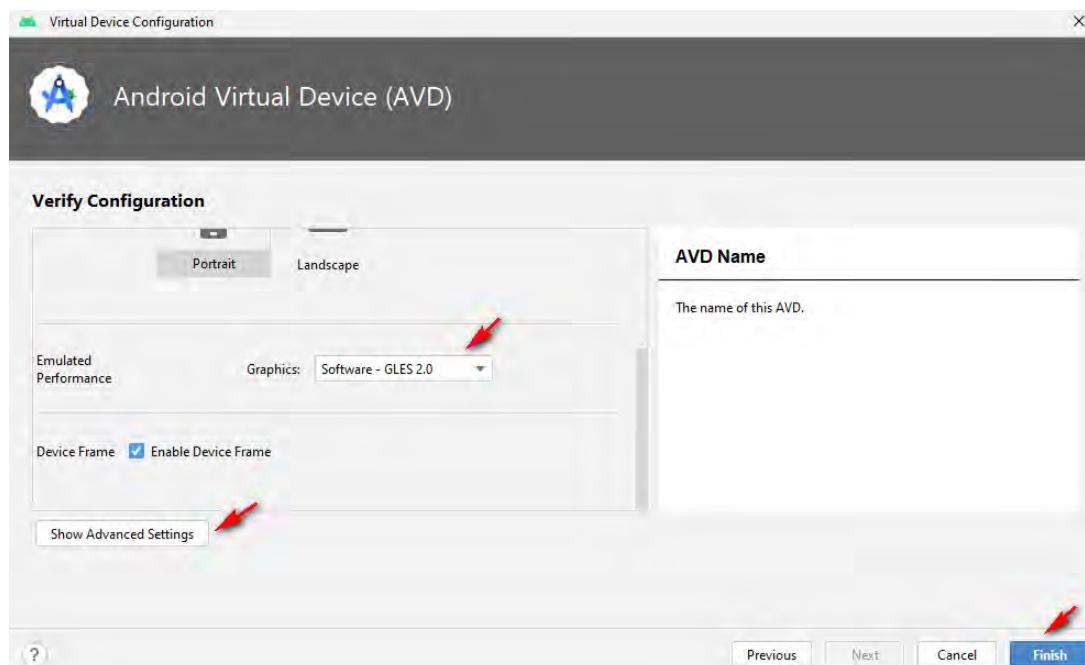
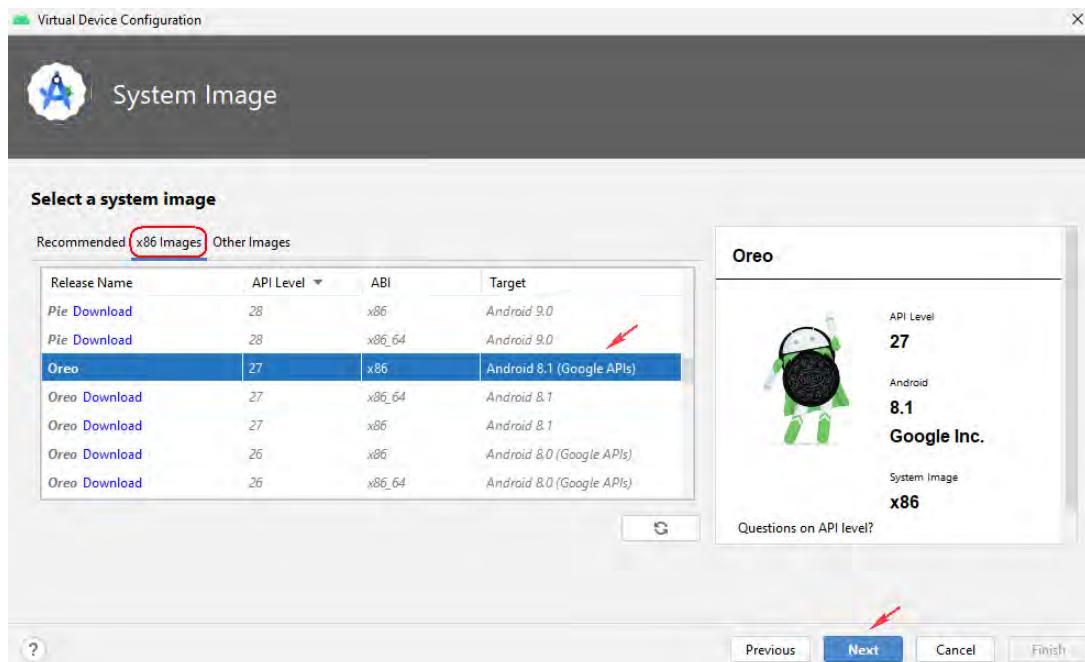
1. Click the “More Actions” dropdown list and select “Virtual Device Manager”. Follow the steps shown in the subsequent screenshots:



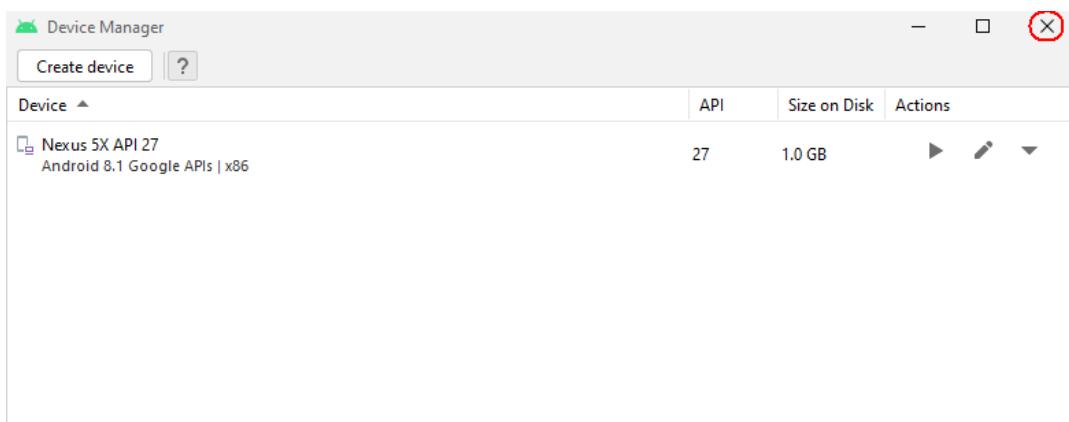
No virtual devices added. Create a virtual device to test  
applications without owning a physical device.  
[Create virtual device](#)



Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering



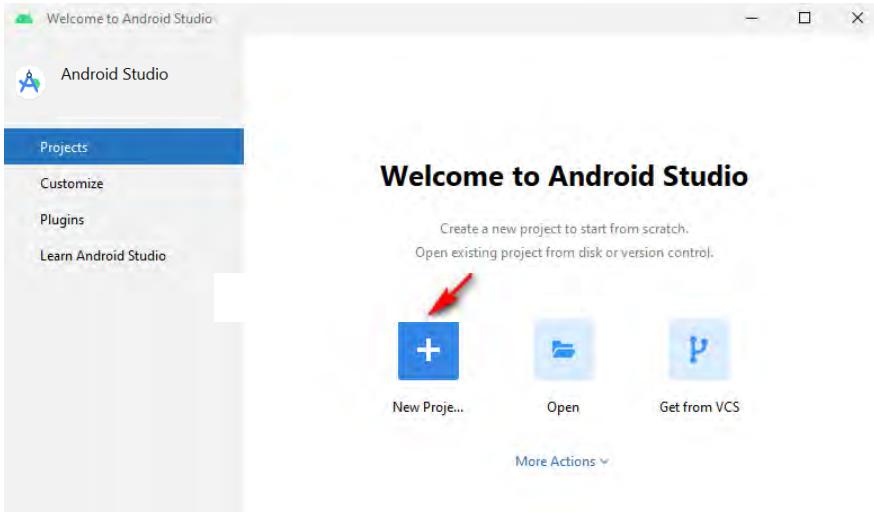
2. Close the "Your Virtual Devices" dialog prompt.



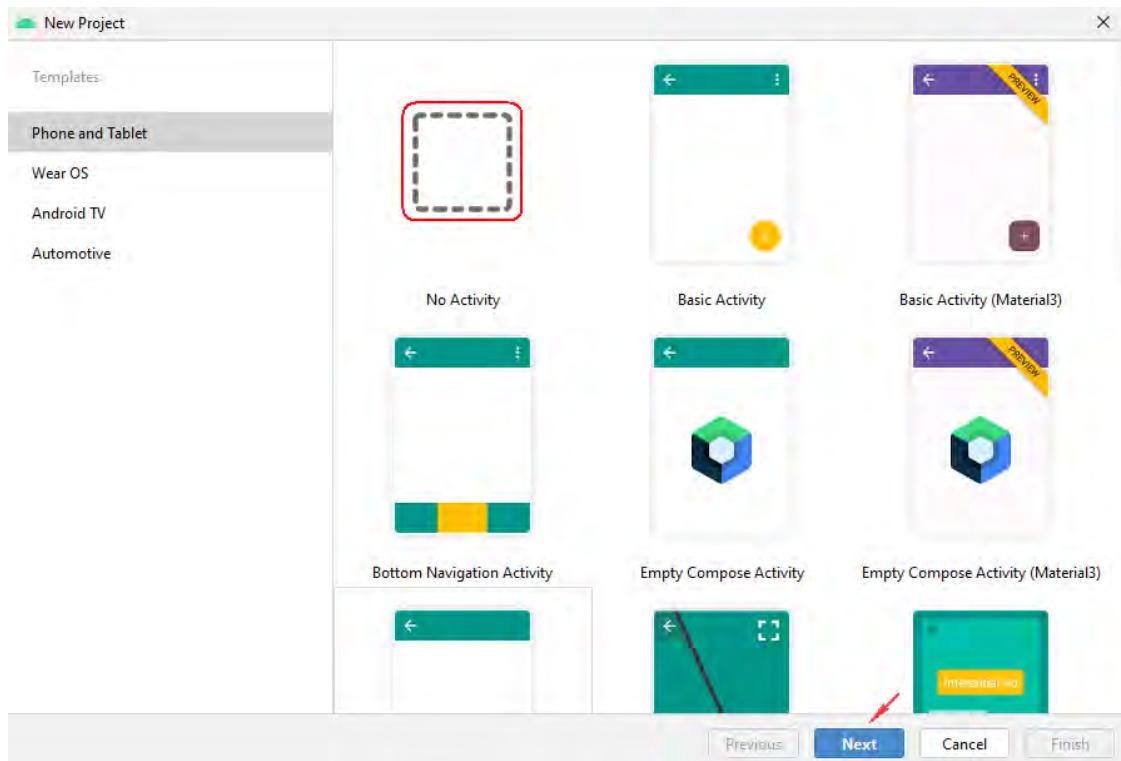
**Note:**

1. You might need to enable “Intel Virtual Technology” for your notebook.
2. Google “How to enable Intel Virtualization Technology” for your brand and model of your notebook as the procedures are different for each brand and model of notebook.

## D. Create Android Test Project

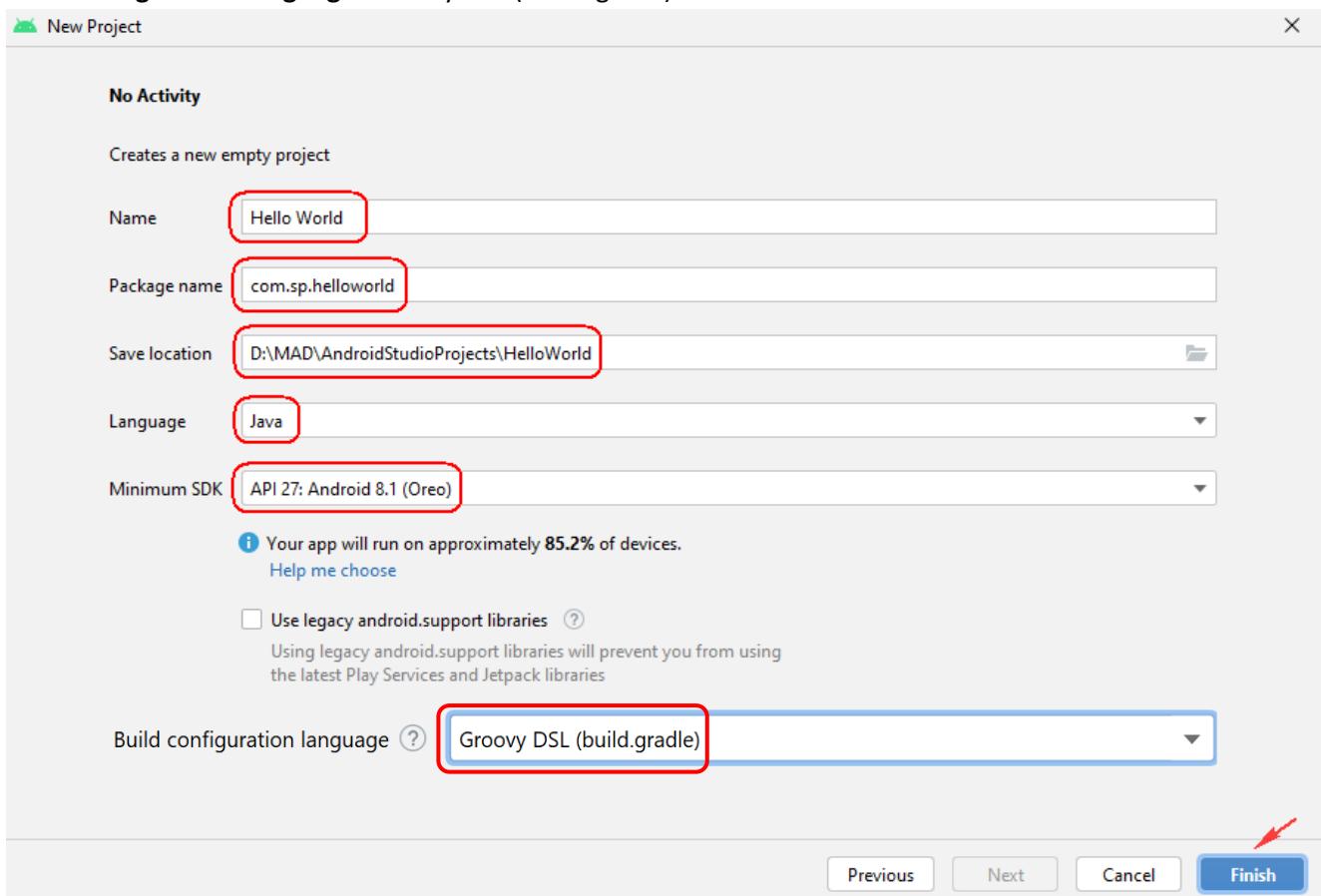


1. To check if the Development Environment has been setup properly, you will need to **start a new Android Studio Project**

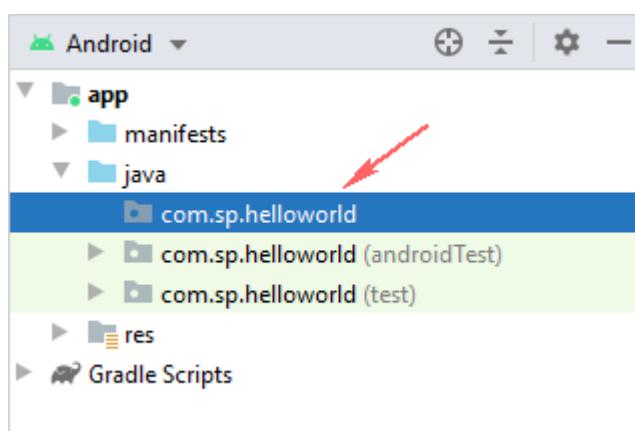


2. Enter the following information for your new Android Application Project

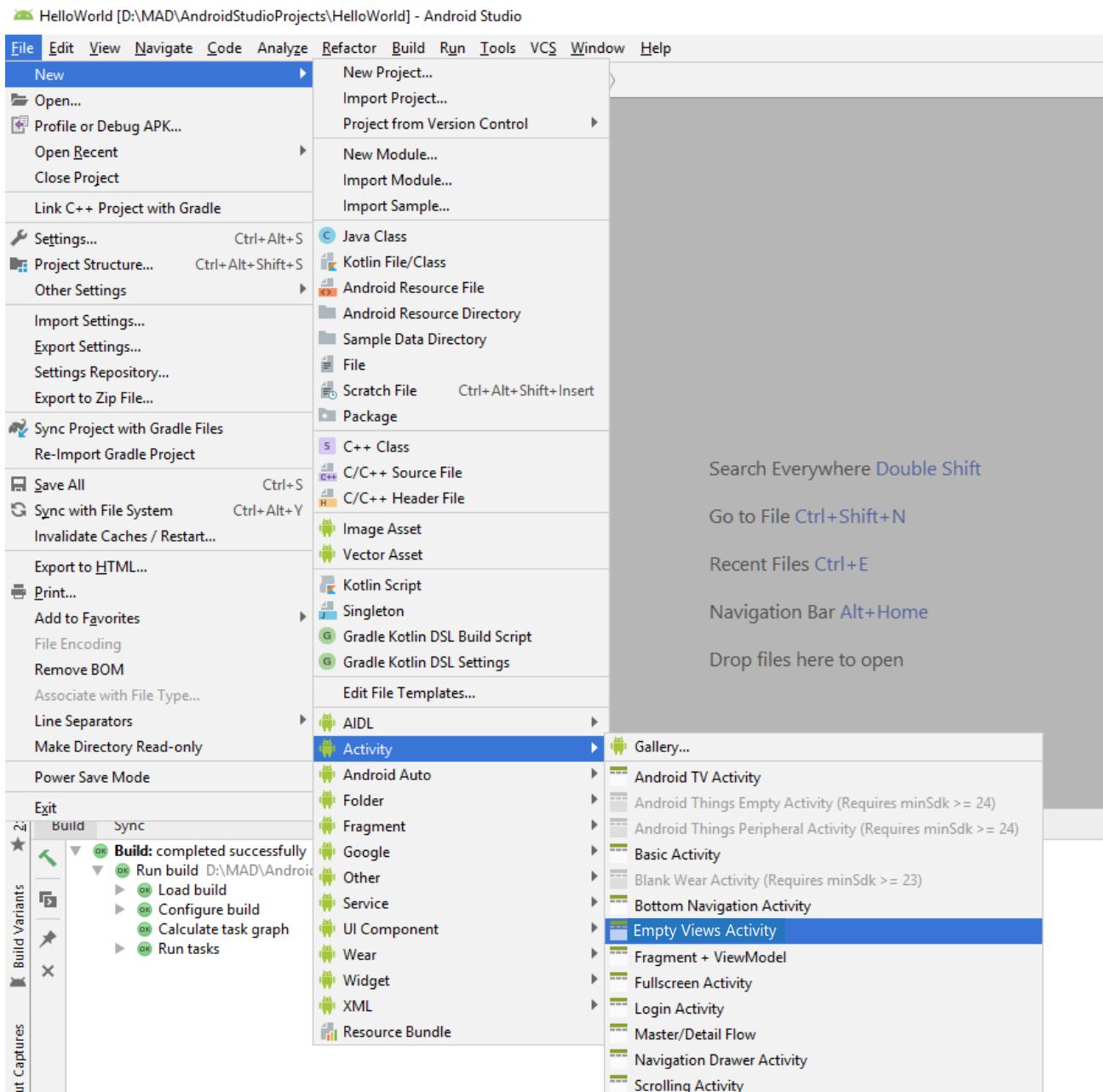
- **Name :** *Hello World*
- **Package Name:** *com.sp.helloworld*
- **Save Location :** *C:\MAD\AndroidStudioProjects\HelloWorld* or *D:\MAD\AndroidStudioProjects\HelloWorld*
- **Language:** *Java*
- **Minimum SDK:** *API 27: Android 8.1 (Oreo)*
- **Build configuration language:** *Groovy DSL (build.gradle)*



3. For the first project you have just created, the IDE will take some time to sync some files remotely. So be patient.  
4. Expand the “app” node and “java” node. Select “com.sp.helloworld”.

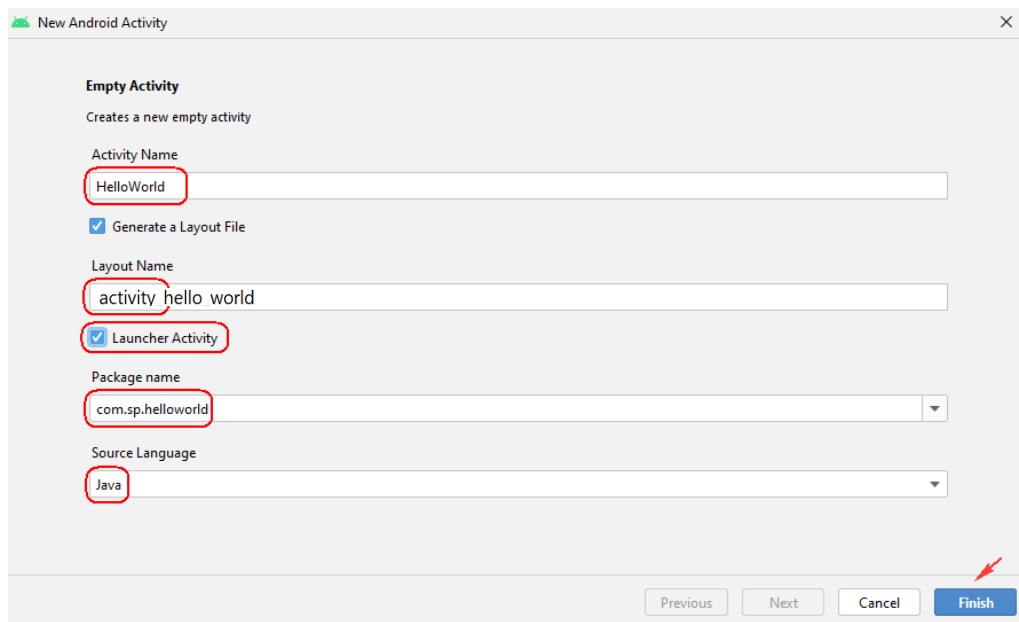


5. Now you will add an “Empty Views Activity” to the project.

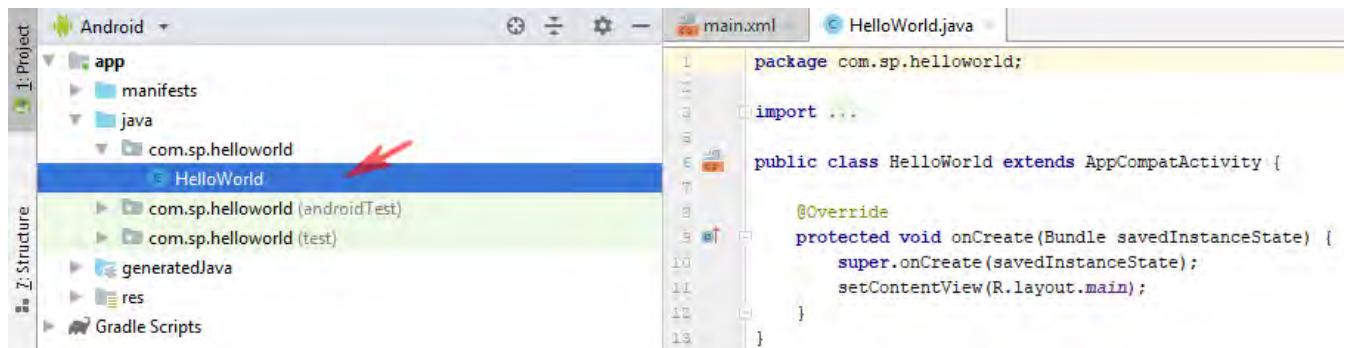


6. Enter the following information for your new Android activity.

- **Activity Name:** *HelloWorld* (*Note: Activity name must be a whole word. No space is allowed*)
- **Layout Name:** *main* (*Note: Layout name must be a whole word. No space is allowed*)
- **Launcher Activity:** *checked* (*Note: Check this option if this activity is the start-up activity, otherwise, uncheck*)
- **Package name:** *com.sp.helloworld*
- **Source language:** *Java*

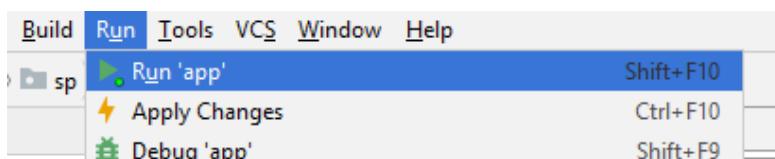
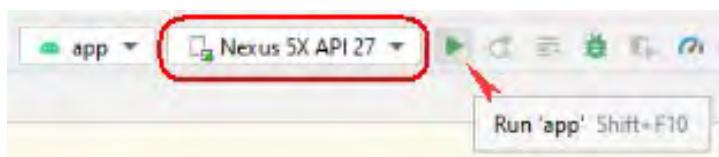


- Notice that HelloWorld activity is created under the “com.sp.helloworld” package.

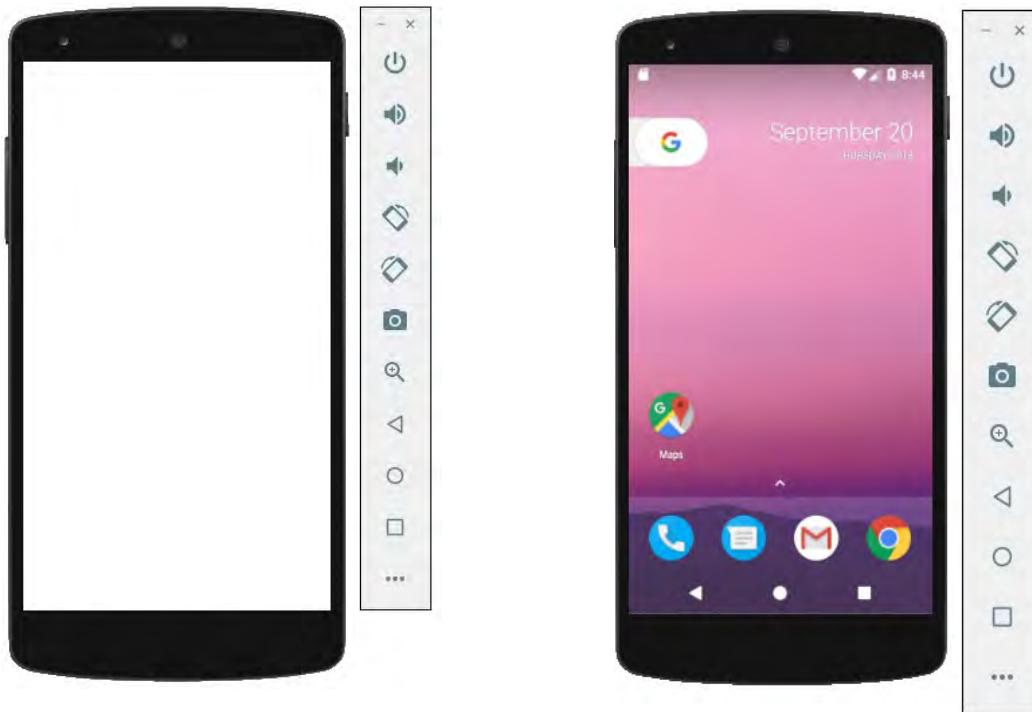


## E. Deploy App to Android Virtual Device

- At IDE top menu, on the drop-down list select **Nexus 5X API 27** and click the “Run” icon or via the “Run” menu. Be patient with the start-up of the emulator.



In case the emulator displays a transparent screen as shown below (left), **resize** your emulator and you should see a normal emulator (right).

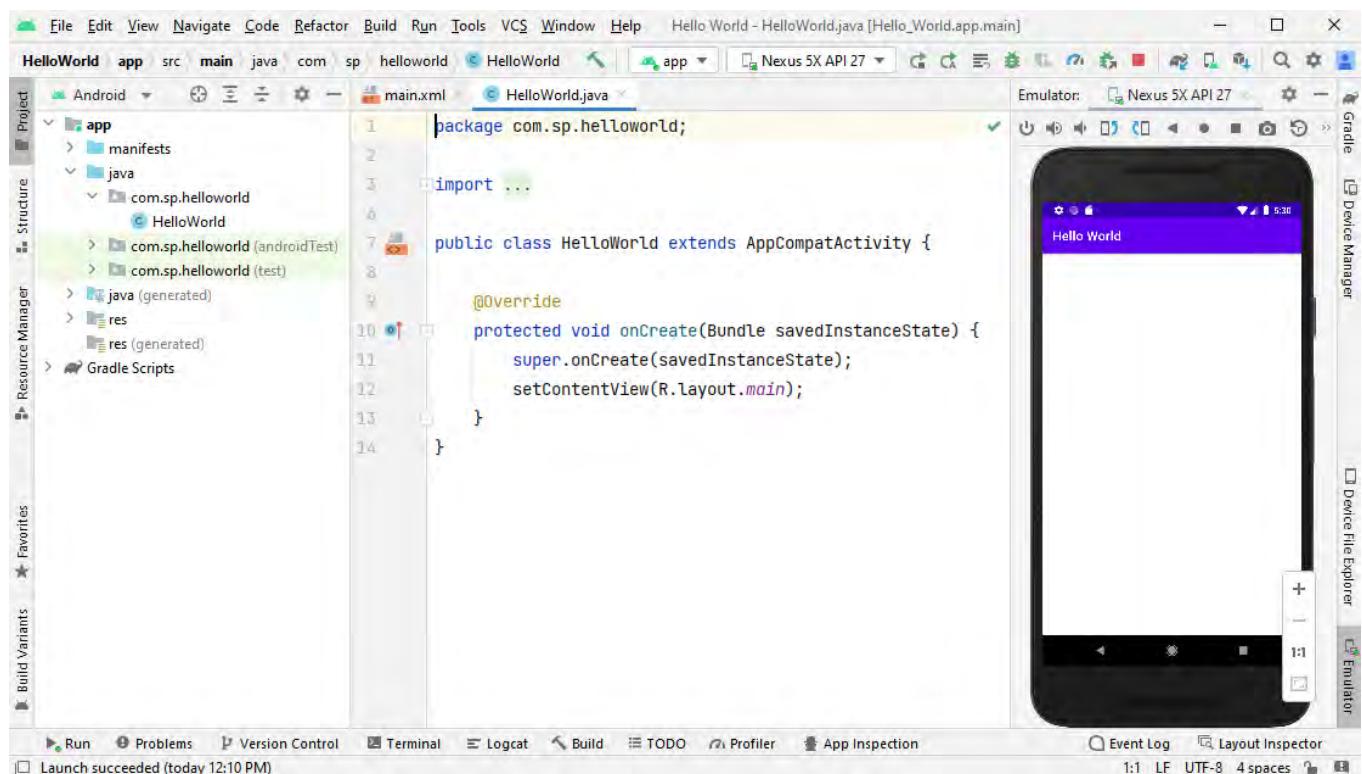
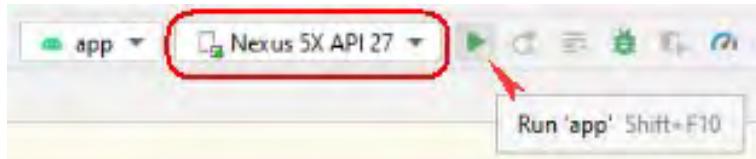


2. Upon successful deployment, the emulator output is as shown below.



## F. Run the Android Emulator directly in Studio

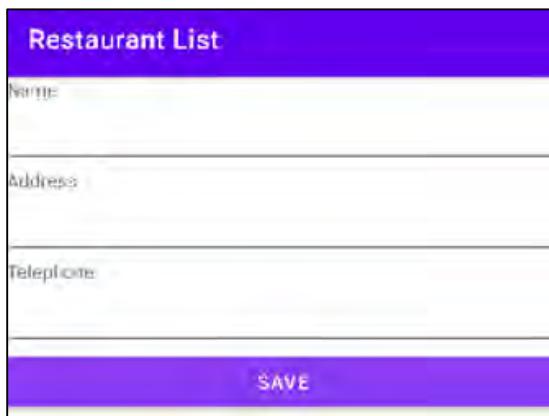
1. You can run the Android Emulator directly in Android Studio.
2. Follow these steps:
  - a. **On Window** Android Studio: **Click File > Settings > Tools > Emulator**, then select “**Launch in a tool window**” and click OK.
  - b. **On macOS** Android Studio: **Android Studio > Preferences > Tools > Emulator**, then select “**Launch in a tool window**” and click OK.
3. Start your virtual device using the AVD Manager or by targeting it when running your app.



-END-

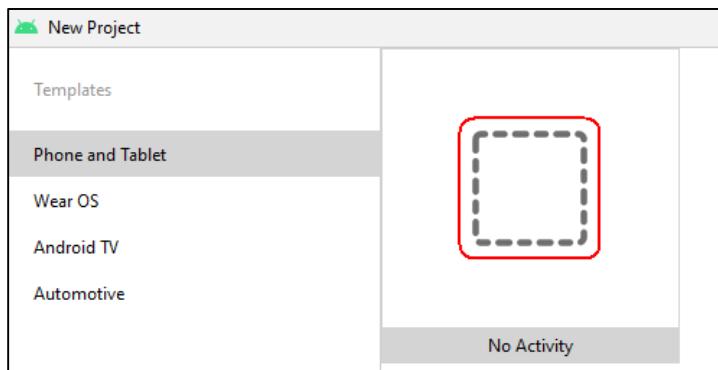
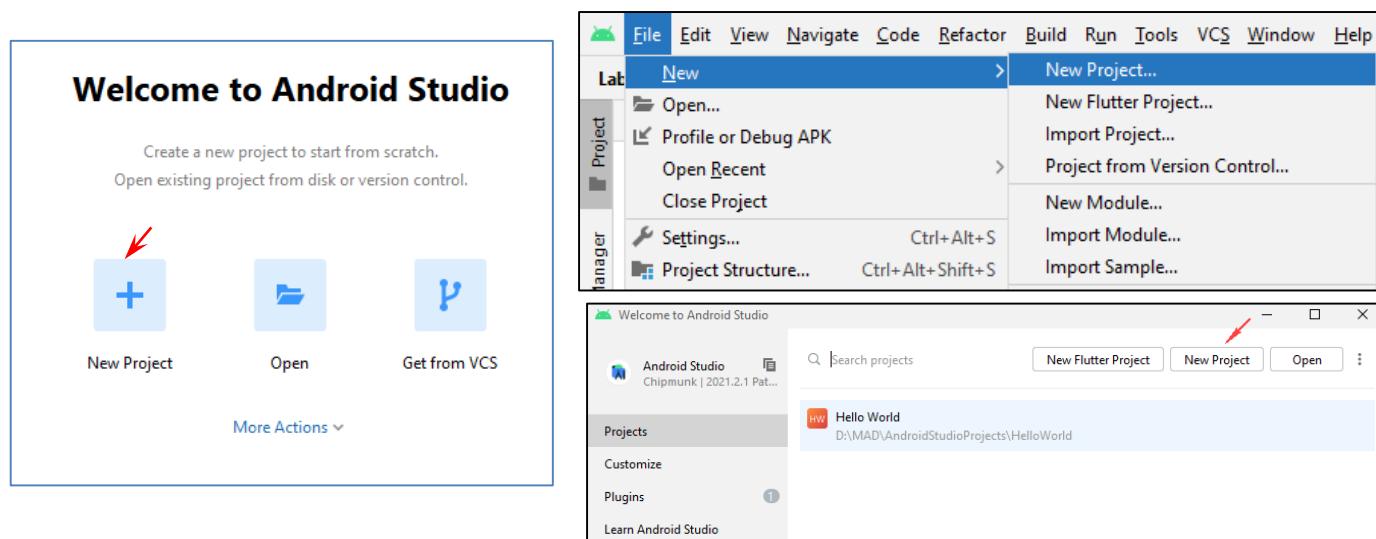
## Practical 1: UI Design

In this session, you will learn how to use Android Layout Editor, both Graphical mode and Text mode, to design a UI (User Interface) as well as setting the attribute of widgets to format data entry



### Part I – Creating an Android Application Project

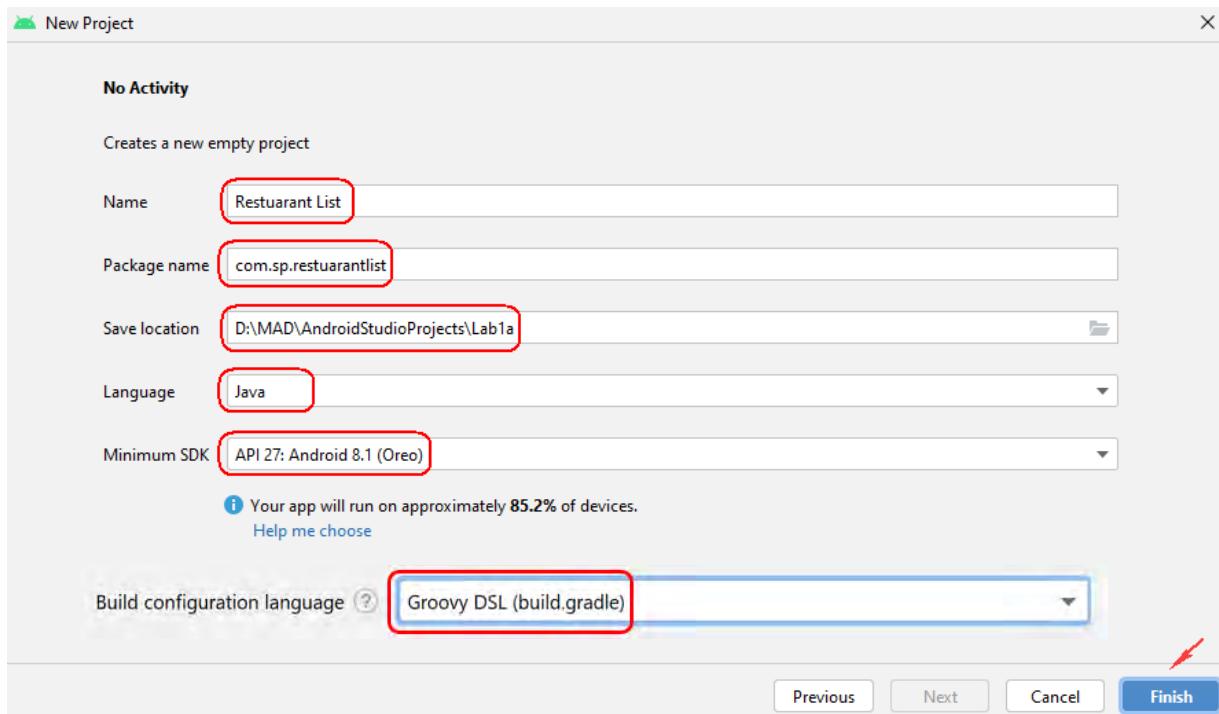
1. Start a new Android Studio project and select the “**No Activity**” project type. Use one of the methods as below.



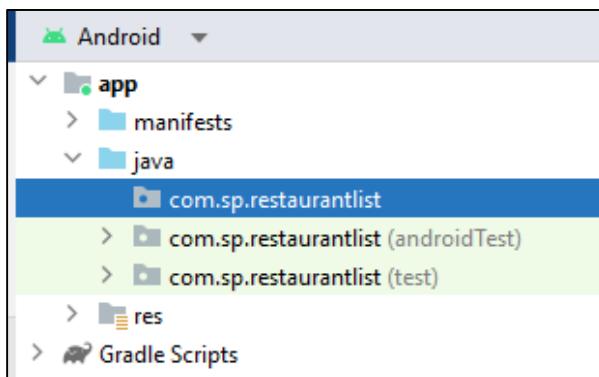
Click Next.

2. Configure your project as follow and click the “Finish” button:

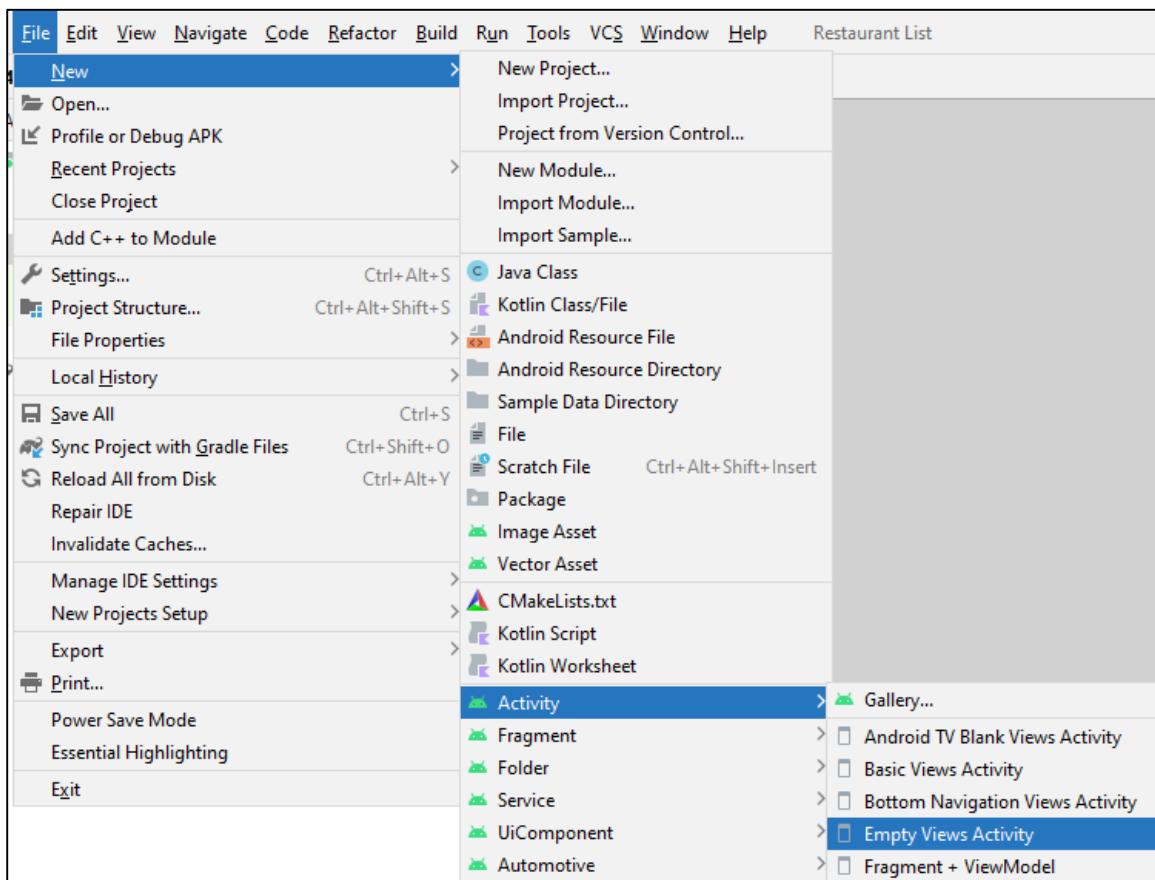
- **Name:** Restaurant List      **(Note: With space in between. This name will appear on the phone screen)**
- **Package name:** com.sp.restaurantlist
- **Save location:** D:\MAD\AndroidStudioProjects\Lab1a or C:\MAD\AndroidStudioProjects\Lab1a
- **Language:** Java
- **Minimum SDK:** API 27: Android 8.1 (Oreo)
- **Build configuration language:** Groovy DSL (build.gradle)



3. Expand the “app” node and “java” node. Select “com.sp.restaurantlist”.

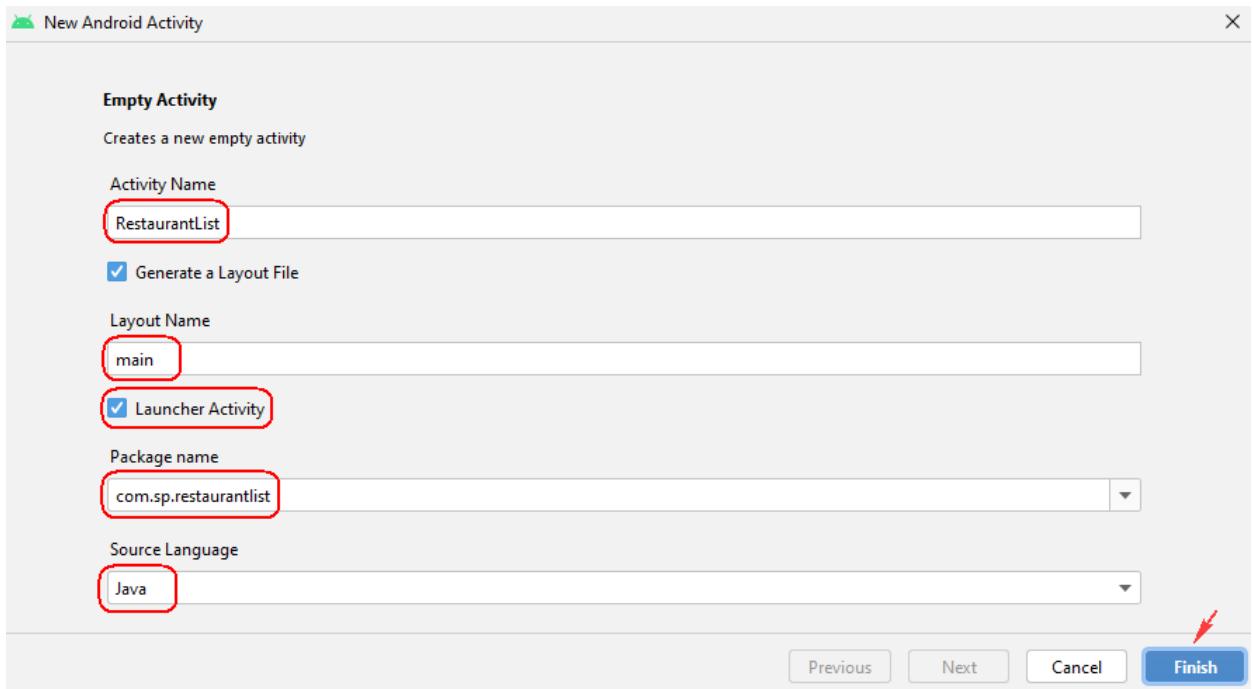


4. Now you will add an “Empty Views Activity” to the project.



5. Enter the following information for your new Android activity.

- Activity Name:** RestaurantList *(Note: Activity name must be a whole word. No space is allowed)*
- Layout Name:** main *(Note: Layout name must be a whole word. No space is allowed)*  
*(Note: This layout file will bind to RestaurantList activity as a user View on the phone display)*
- Launcher Activity:** Check *(Note: Check this option as this activity is the start-up activity)*
- Package name:** com.sp.restaurantlist
- Source language:** Java



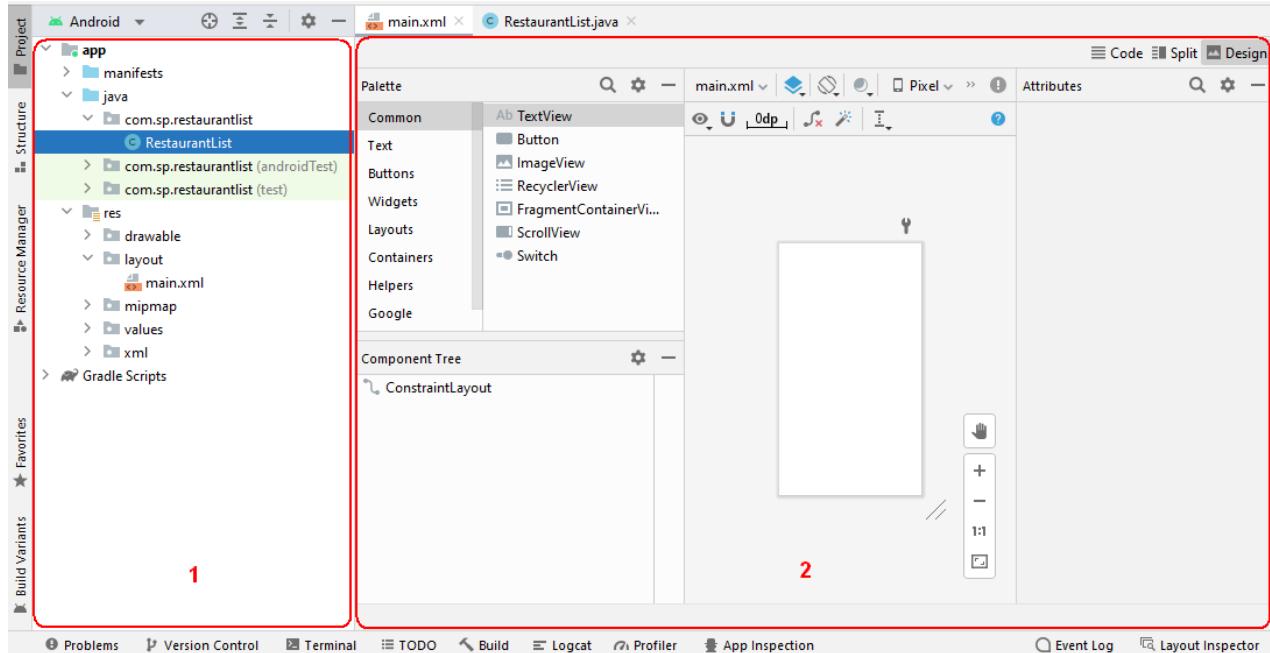
6. Lab1a project will be created as shown

The screenshot shows the Android Studio interface. On the left, the project structure is displayed under the 'app' folder. It includes 'manifests', 'java' (containing 'com.sp.restaurantlist' which has 'RestaurantList' selected), 'res' (containing 'drawable', 'layout' with 'main.xml', 'mipmap', 'values', and 'xml'), and 'Gradle Scripts'. In the center, the code editor shows 'RestaurantList.java' with the following content:

```
package com.sp.restaurantlist;  
import ...  
2 usages  
public class RestaurantList extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
}
```

7. Take a look at the Android Studio IDE view.

- View 1 – Project View showing the contents of the project structure
- View 2 – Editor View to allow user to edit User Interface (UI) layout and program (select tab **main.xml**)

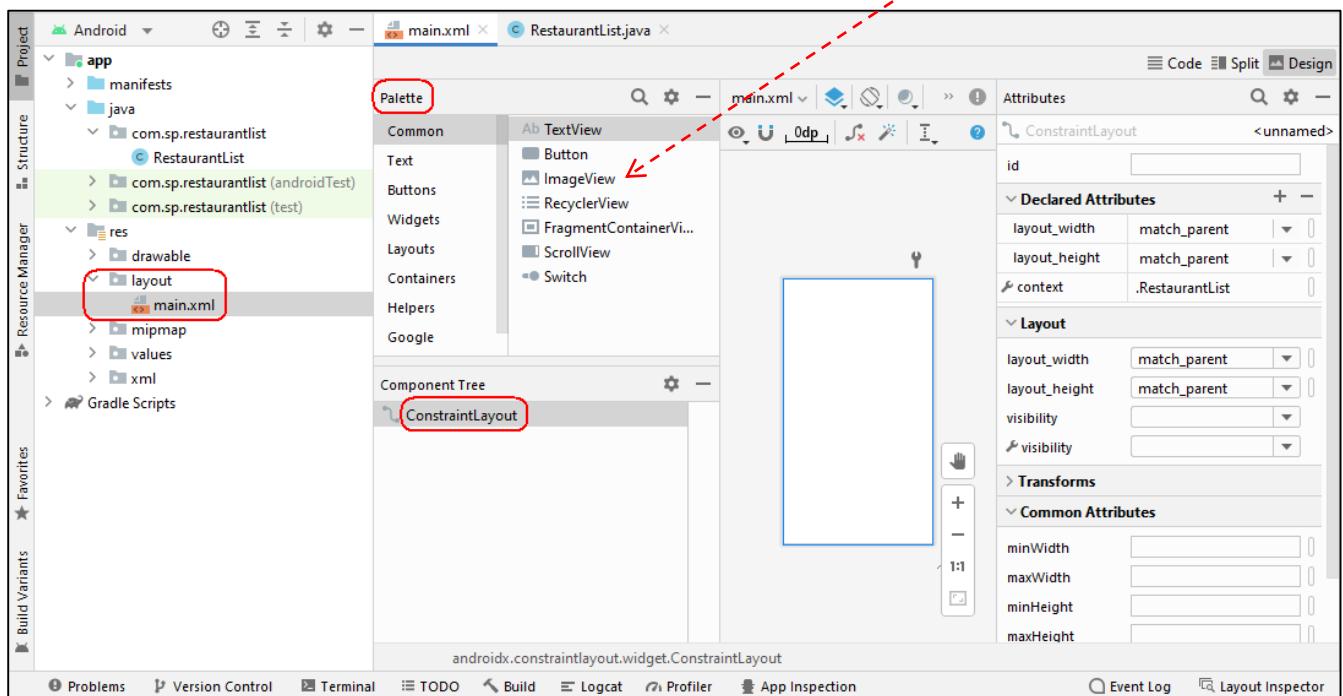


8. At the Layout Editor, the **main.xml** file is the layout file created for **RestaurantList.java** Activity

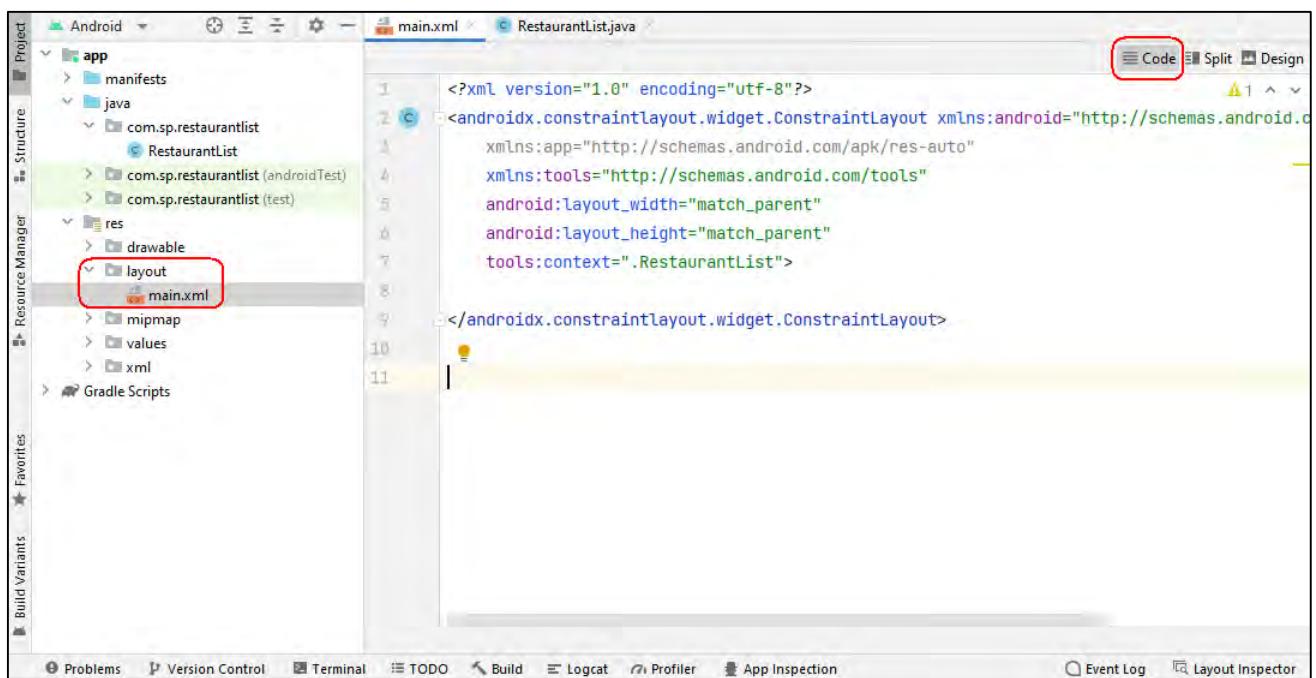
9. By default, all layout files shall be stored in the **res/layout** folder.

10. The IDE has provided to two options of **Layout Editor** to allow user to create User Interface (UI) View:

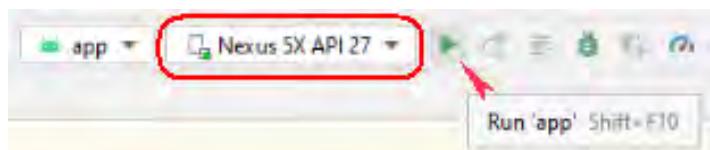
- **Design Editor** – allow user to drag and drop widgets from **Palette** into the **Design Editor** view or into **Component Tree**



- Text Editor – create UI View using XML text file  
(Note to indent the file, at Top MENU Bar, click on **Code > Reformat Code** at the top menu bar)



- To run the app created in the *Lab1a* project, at the top menu bar and on the drop-down list select **Nexus 5X API 27** and click on button or via the “Run” menu.

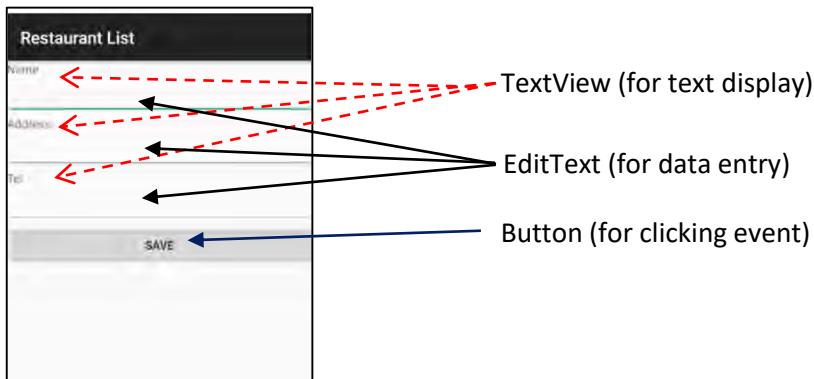


- If successful, you will see an emulator launched.

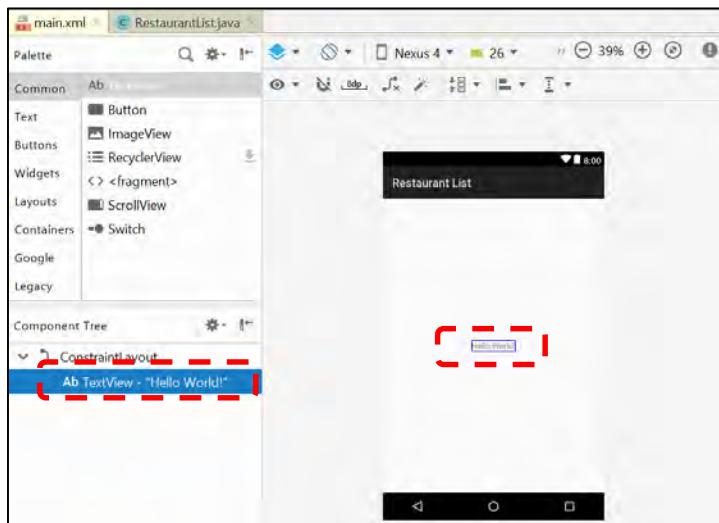


## Part II - Creating Simple Form UI View

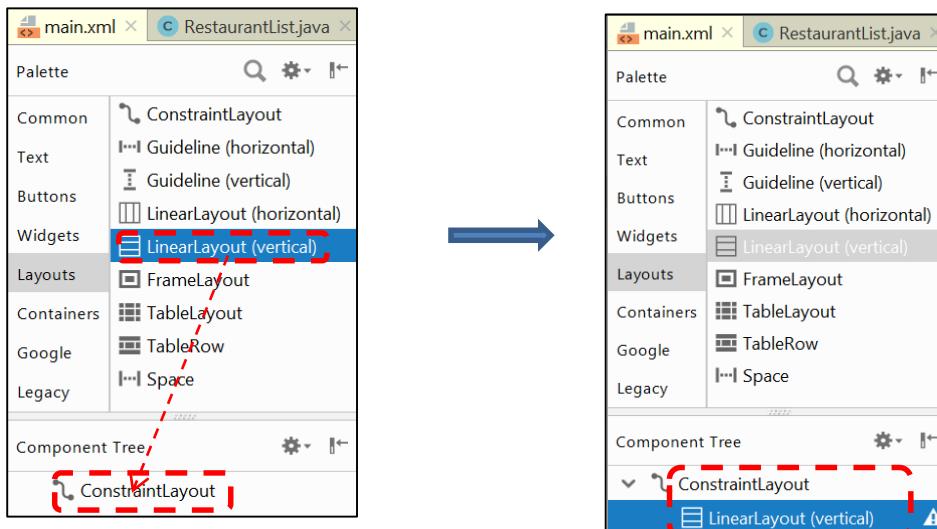
13. The subsequent steps will lead you to create a new layout for the *Restaurant List* application. It comprises three *TextView*, three *EditText* and a *Button* widgets



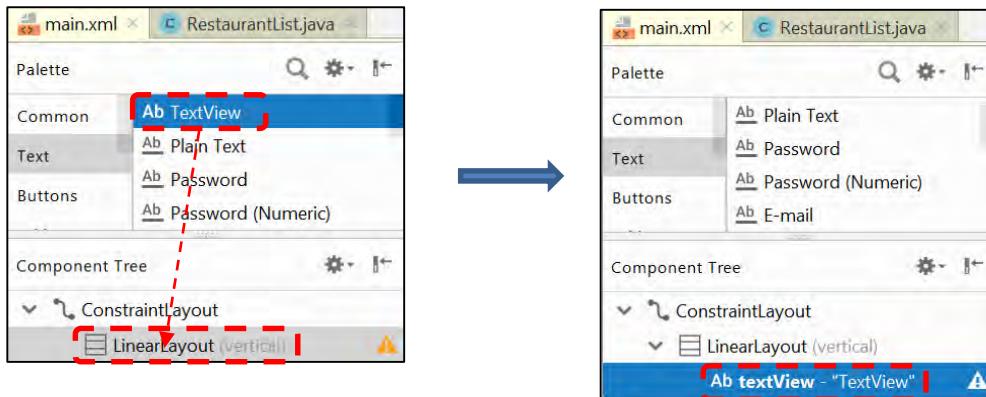
14. At the Design Editor pane of **main.xml** file, if you have any other widgets, delete them. For example if there is a *TextView* widget, click on the *TextView* widget in the Component Tree and press delete key from keyboard to remove the widget from the View. (You may need to expand the Palette pane if it appears on the side as a vertical text)



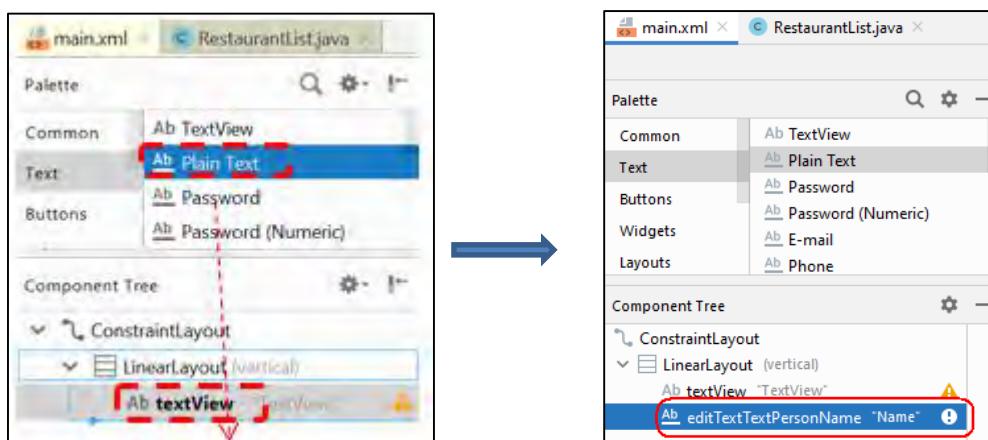
15. Select the *LinearLayout (Vertical)* from **Palette pane**. Drag and drop it onto the *ConstraintLayout* in the **Component Tree**



16. At Palette pane, drag **TextView** component from **Text** option and drop into **LinearLayout**

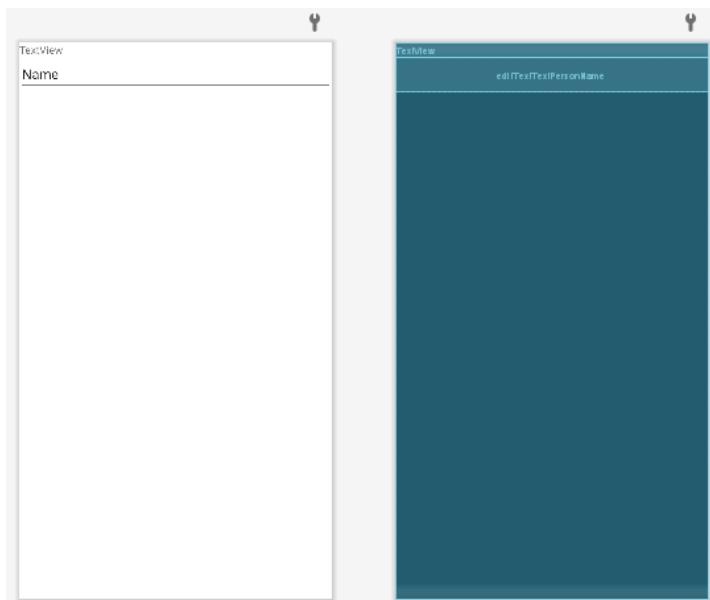


17. Drag a **Plain Text** component from **Text** option and drop below **textView** component



18. What you have done so far is to create a UI (user interface) display using components provided

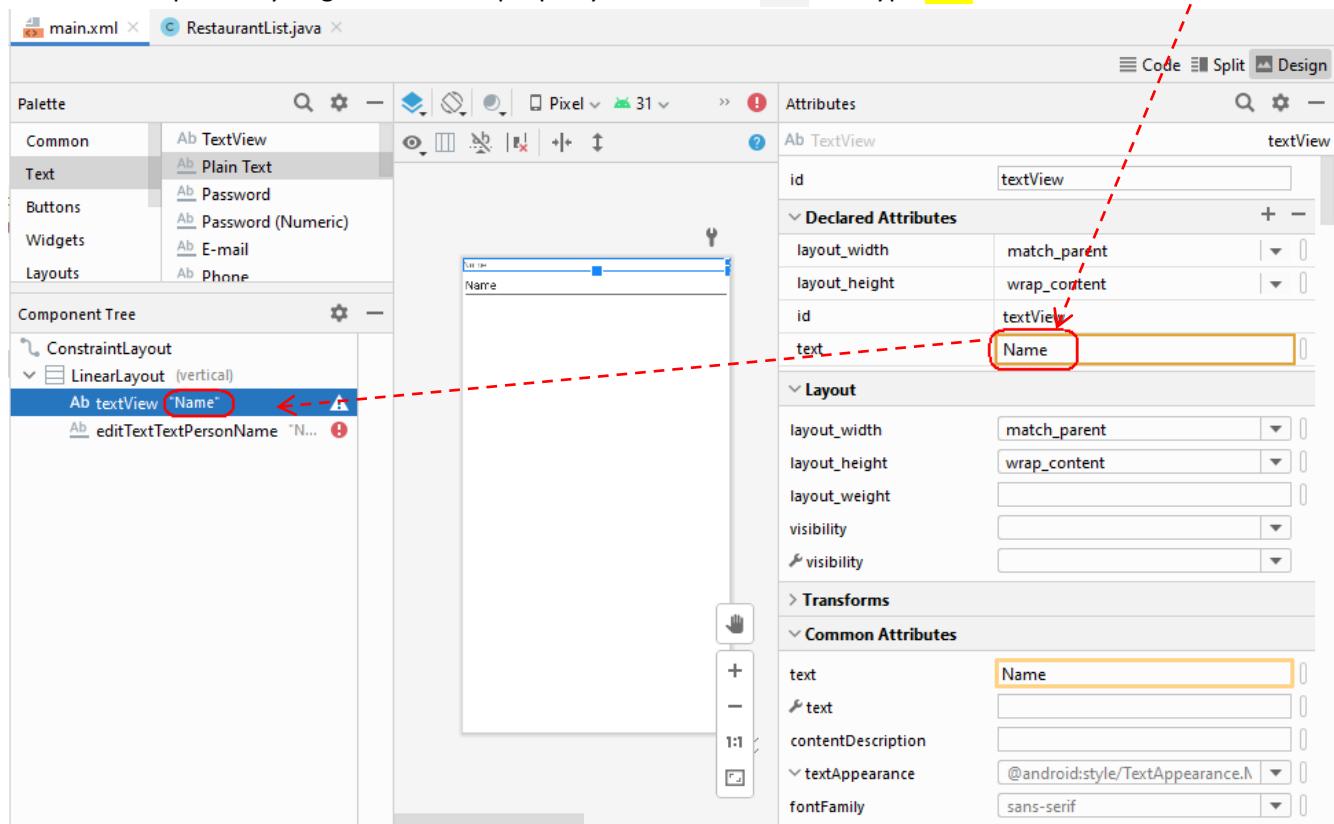
- *TextView* Widget – to display information on user screen
- *EditText* Widget – to allow user to enter information



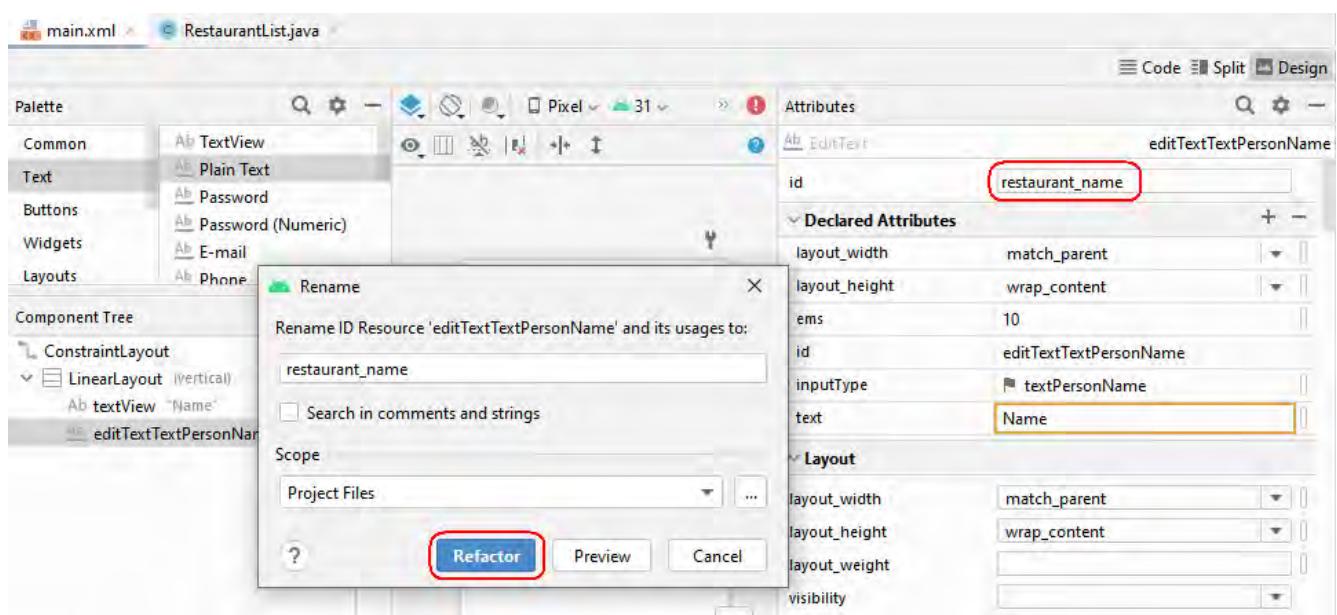
19. As can see from the *Design Editor View*, the **label** displayed on screen is **not correct**. The intention of *TextView* widget in this form is to ‘show’ user that the following data entry (*EditText* widget) is for ‘Restaurant’s Name’. To edit the values, you will need to update the ‘Properties’ of these widgets. To make

changes on these values, you will need to click on the individual widgets and update the content of the widget's properties

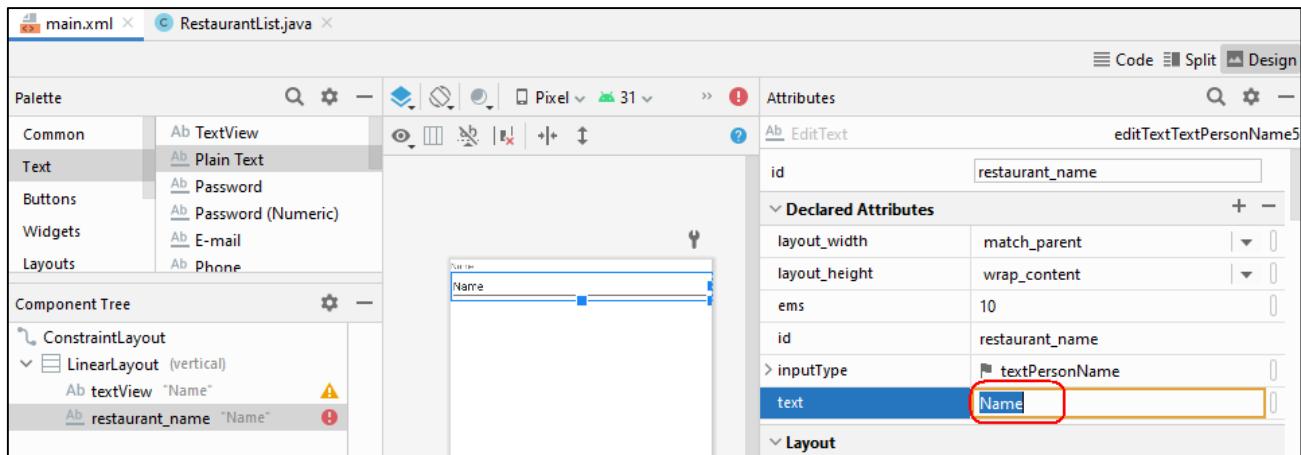
20. To change the **TextView** widget to show 'Name' as label, click on the **textView** widget at the **Component Tree panel**. At the **Attributes** panel (on the right hand side of the *Design Editor View*), update the **text** value to 'Name'. A quick way to get to the text property is to click on and type **text**.



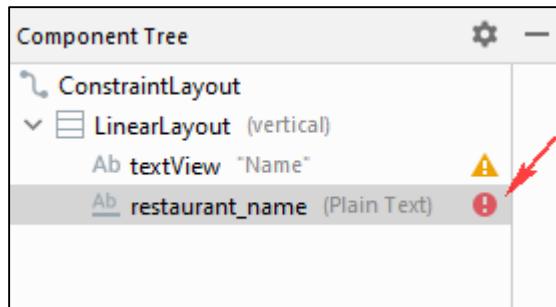
21. To change the **EditText** widget to display blank entry by default and the **widget ID** to '**restaurant\_name**', click on the **EditText** widget at the **Component Tree panel**. At the **Attributes** panel, update the **ID** to '**restaurant\_name**'. At last, click the "**Refactor**" button.



22. Double click the **text** property and clear the content to blank.



23. Notice if there is a red error symbol next to the “restaurant\_name” widget.



To remove these errors:

1. For the “hint” attribute enter a hint text e.g. “Enter restaurant name”

hint

2. Set the “minHeight” attribute to 48dp or more.

minHeight

Otherwise, you can also ignore these errors.

24. Repeat **Step 16 to 23** to add 5 more widgets to the **Component Tree** of *main.xml* layout with the following property settings:

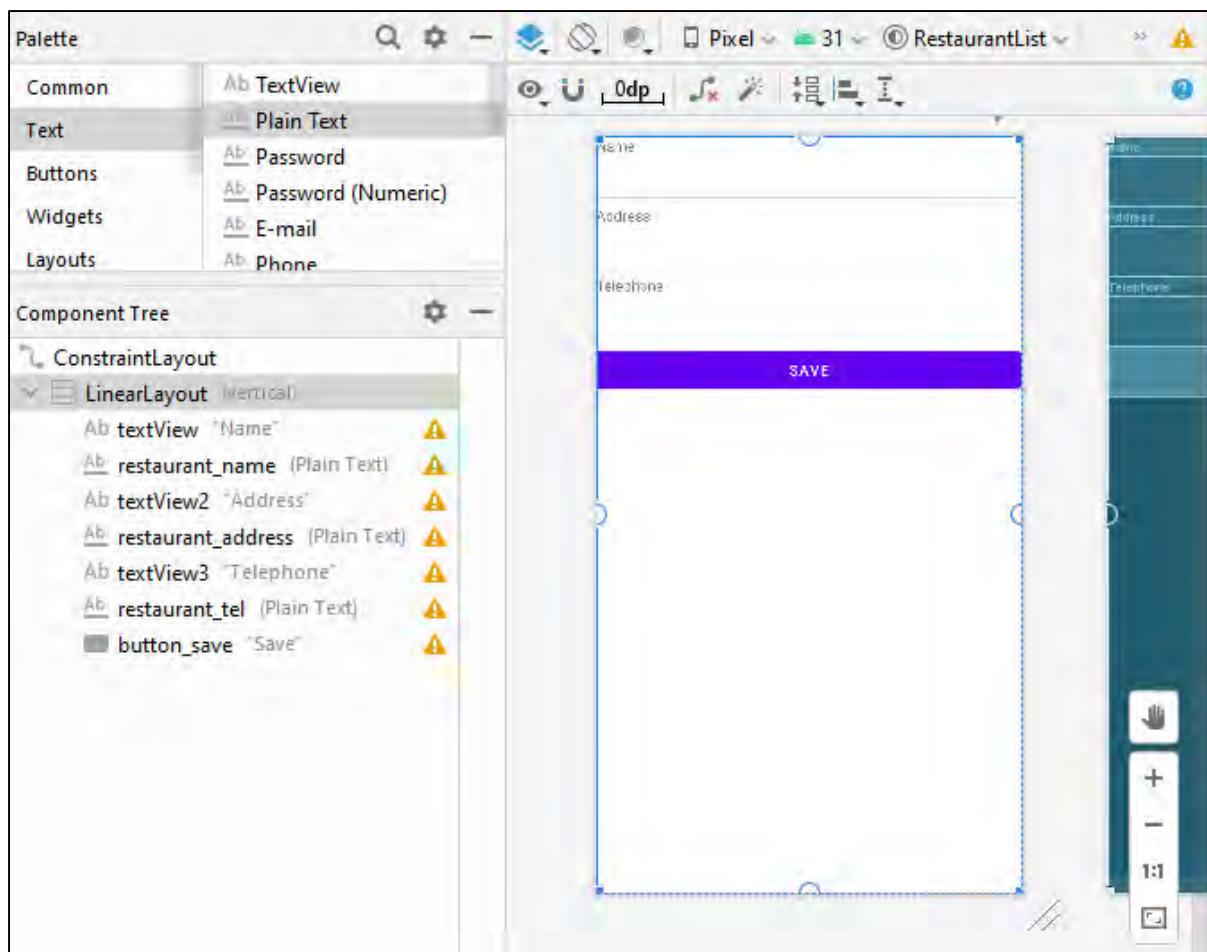
| Widget   | text property | id property        | remarks                                 |
|----------|---------------|--------------------|-----------------------------------------|
| TextView | Address       | (not applicable)   |                                         |
| EditText | (blank it)    | restaurant_address | For user to enter address of restaurant |
| TextView | Telephone     | (not applicable)   |                                         |
| EditText | (blank it)    | restaurant_tel     | For user to enter tel# of restaurant    |
| Button   | SAVE          | button_save        | For user to click                       |

A quick way to get to the any property is to click on and type in the property.

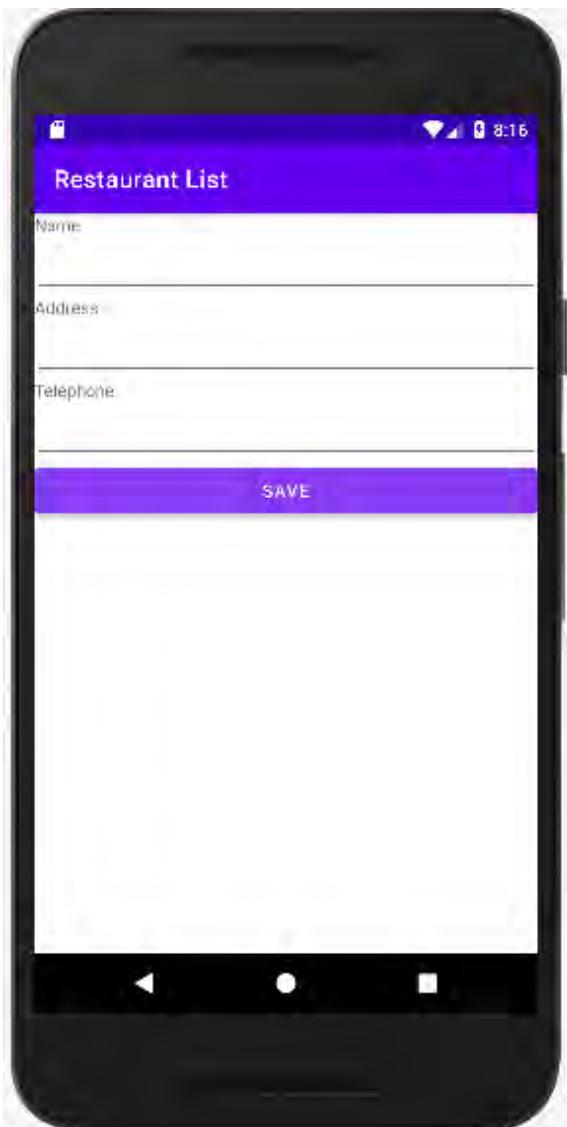
You can also adjust the textSize property to your liking.

Every property id has to be unique. It cannot be duplicated.

Pay attention to the correct spelling of all the ids. These are to be used later in coding.



25. Click on button at the top menu bar to run the app
26. If successfully, an emulator will be launched and the form designed will be loaded. It will take quite some time for the emulator to run. Please be patient.
27. This simple UI View allows user to enter information by clicking the respective input box and type in the text.



### Further Enhancements

28. If you notice, the input fields (*EditText* widgets) have no constrained on the input data type or length! For instance, at the moment, users are not stopped from entering alphabet letter as telephone number. The control can easily be done through adding extra attribute to the respective *EditText* widgets in *main.xml* layout. Please modify the properties of the following *EditText* widgets

- ‘*restaurant\_name*’ – maximum 30 characters
- ‘*restaurant\_address*’ – maximum 60 characters
- ‘*restaurant\_tel*’ – maximum 8 digits number

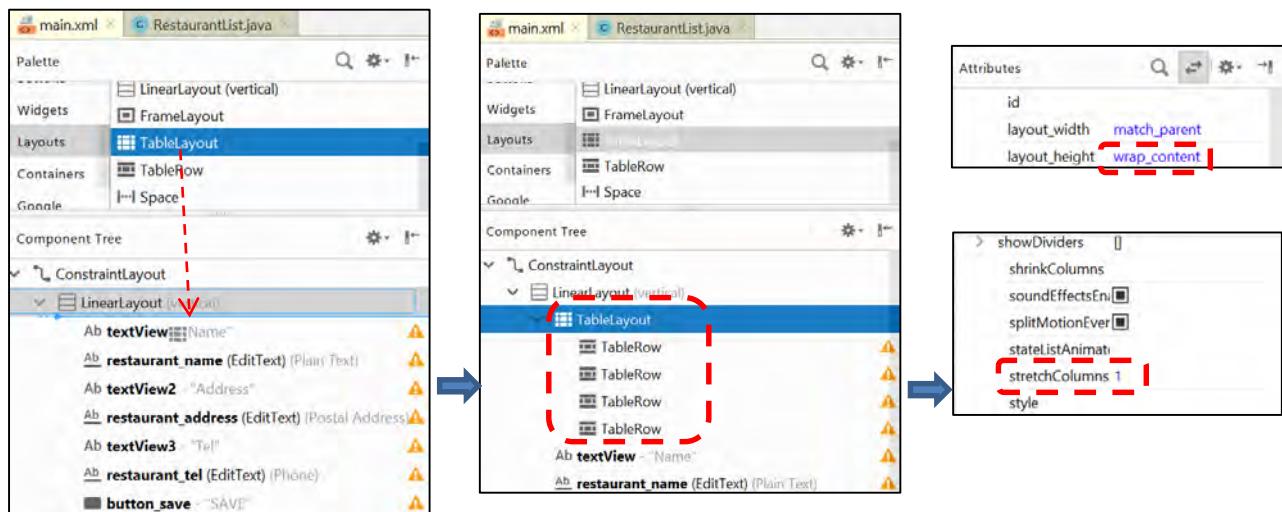
**Hint:** You can edit the **Maximum Length** attribute display in **Properties** pane.

(**Note:** You do not have to close the emulator. For every new run of your Android application, the system will overwrite the old application in the emulator)

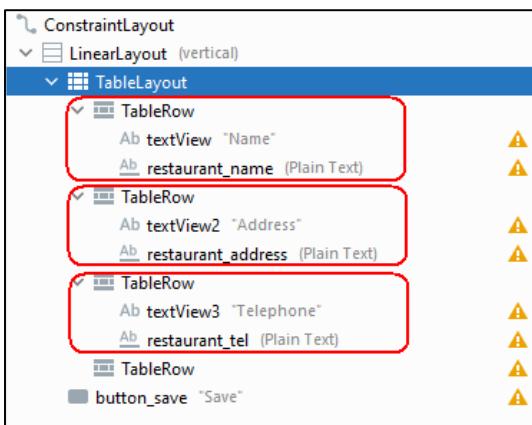
29. **Show your lecturer after you have achieved the Further Enhancements.**

### Part III – Creating a Fancier UI View

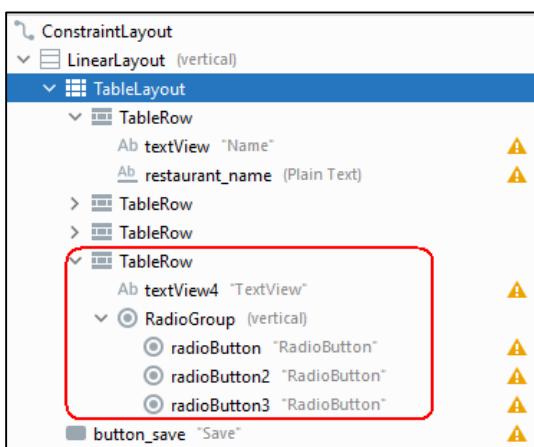
- In this section of the exercise, *TableLayout* will be used to enhance the look and feel of the previous form design. The *TableLayout* allows user to assign widgets in rows and columns of a table. Here, the table will display rows of widgets in the form of a column. It looks better as comparing to *LinearLayout*
- At **Component Tree**, drag a *TableLayout* widget and drop into *LinearLayout* as shown below. Click on *TableLayout* widget and update the **layout\_height** to ‘wrap\_content’ and **stretchColumns** property to ‘1’ at the **Attributes** panel. (To search attribute **stretchColumns**, click on , type “**stretchColumns**”)



- Drag and drop all the *TextView* and *EditText* widgets into the *TableRow* as shown

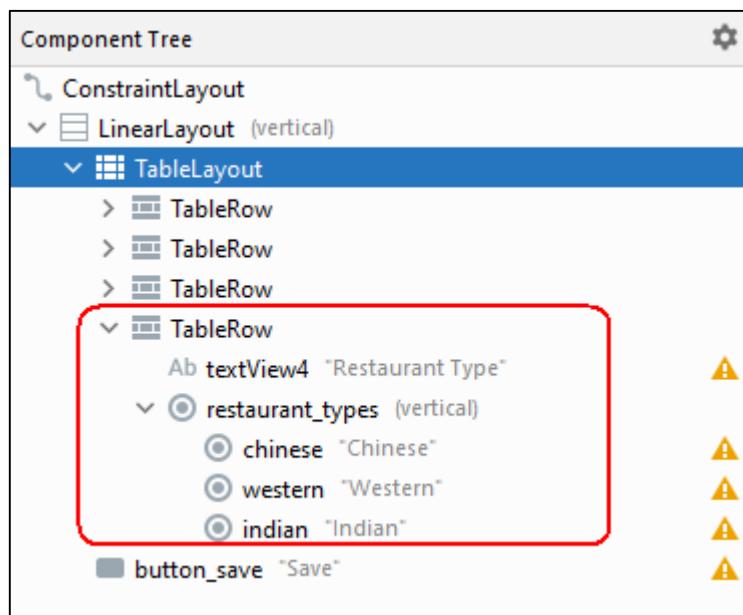


- Drag and drop **one** *TextView* widget and **one** *RadioGroup* widget (from *Buttons* option) into the 4<sup>th</sup> empty *TableRow*. Drag **three** *RadioButton* widgets into the *RadioGroup* widget.

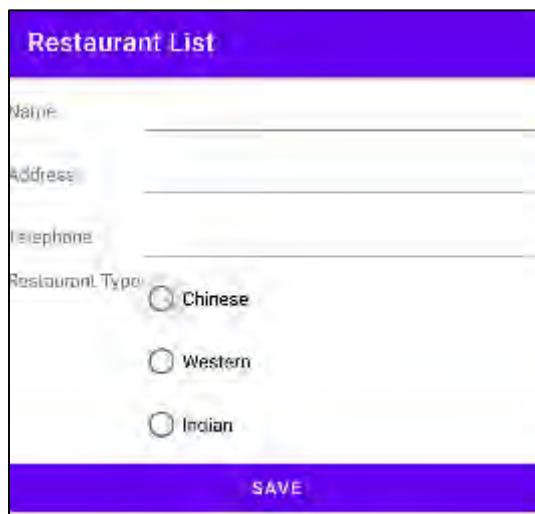


5. Notice if there is a **red error symbol** next to the “radioButton” widget. Double click the “radioButton” widget and for the **“Render” error, click the “Fix” error.**
6. Do the necessary update on the display text and IDs of *TextView*, *RadioGroup* and *RadioButton* widgets as below
  - i. *TextView* – change **text** to “Restaurant Type:”
  - ii. *RadioGroup* – change **id** to “**restaurant\_types**”
  - iii. *RadioButton1* – change **ID** to “**chinese**” and display text to “**Chinese**”
  - iv. *RadioButton2* – change **ID** to “**western**” and display text to “**Western**”
  - v. *RadioButton3* – change **ID** to “**indian**” and display text to “**Indian**”

Pay attention to the correct spelling of all the ids. These are to be used later in coding.

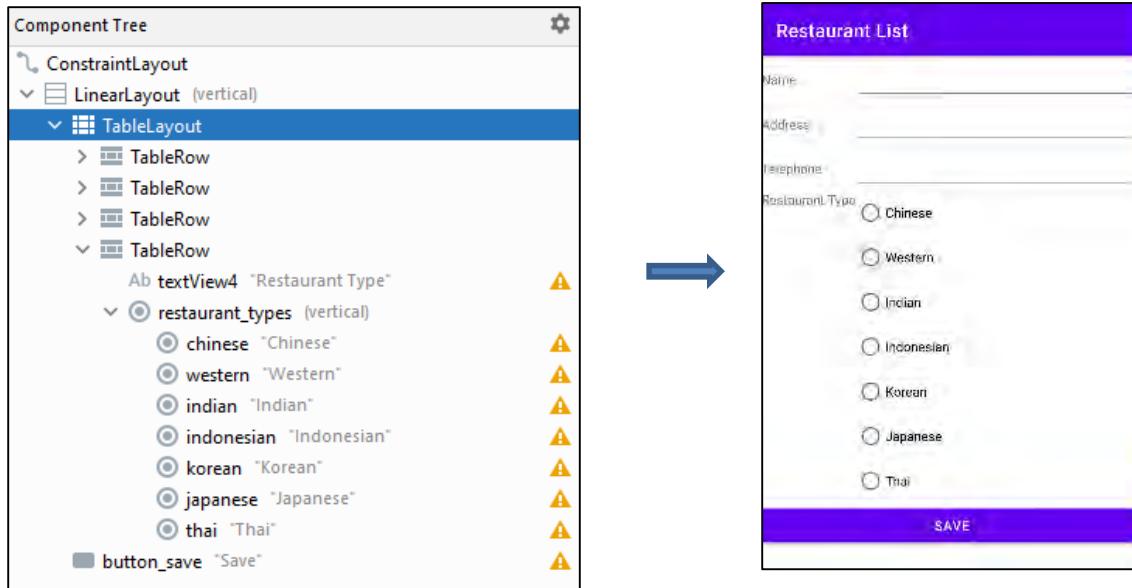


7. Run the Lab1a project. The emulator will load the view as designed



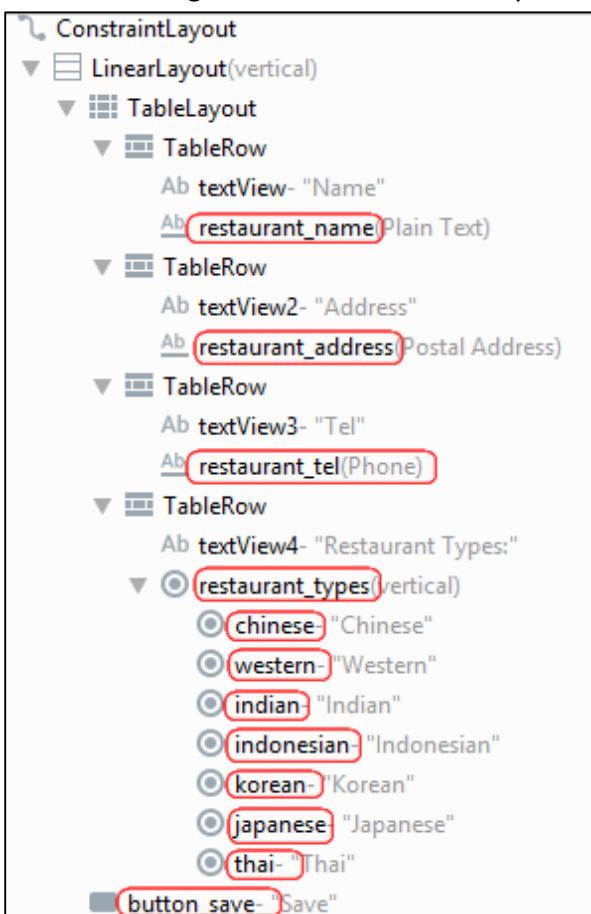
## Further Enhancements

8. Add the 4 more food types (Indonesia, Korean, Japanese and Thai) to the existing *RadioGroup* widget.  
Update the ID and display text for the new food types as shown below. Save and run the app



## Part IV – Getting Data from Form & Detecting Button Click

9. We have created a form UI for user to enter Restaurant data. The subsequent steps will demonstrate how program can respond to “SAVE” button click and read data from *EditText* widget as well as selection of *RadioGroup*
10. Each of the widgets created in *main.xml* Layout have an unique ID



11. In order for program (Controller) to have access to all these widgets, the program must have a variable to bind to them.
12. For example, the following 3 statements allow the Controller to ‘read’ in the restaurant name being entered by the user:

```
private EditText restaurantName;
restaurantName = findViewById(R.id.restaurant_name);
restaurantName.getText().toString();
```

- i. Variables have to be declared before they can be used (e.g. *int marks, char c* in C++).  
 The 1<sup>st</sup> statement declares an *EditText* variable with name as ‘*restaurantName*’:
 

```
private EditText restaurantName;
```
- ii. The 2<sup>nd</sup> statement binds the ‘*restaurantName*’ variable created in the program with the **EditText** widget in the layout (with id ‘*restaurant\_name*’):
 

```
restaurantName = findViewById(R.id.restaurant_name);
```
- iii. The 3<sup>rd</sup> statement calls the *getText()* method (provided by the *EditText* widget) to read in the data entered by the user from the form UI:
 

```
restaurantName.getText().toString();
```

13. To handle the clicking of a button in a program, as the developer, we need to decide what the program should do when the clicking happened. To detect the clicking is not an issue, the system will handle it. The “what-to-do” code or the code that supposed to response to the clicking of the button is what we need to program. It will be housed in a method (similar to function in C++) which will be given a name (just like function in C++).

The following 3 statements handle the clicking of a single button. It will result in calling the method **onSave** whenever the **SAVE** button is being clicked:

```
private Button buttonSave;  

buttonSave = findViewById(R.id.button_save);  

buttonSave.setOnClickListener(onSave);
```

- i. The 1<sup>st</sup> statement declare a *Button* variable with name as '**buttonSave**':

```
private Button buttonSave;
```

- ii. The 2<sup>nd</sup> statement binds the '**buttonSave**' variable created in the program with the *Button* widget in the layout (with id '**button\_save**'):

```
buttonSave = findViewById(R.id.button_save);
```

- iii. The 3<sup>rd</sup> statement directs the program to jump to (or call) the method **onSave** whenever the **SAVE** button is clicked. This process is called registering the '**buttonSave**' button to an Event Listener. In this case, the system is constantly ‘listening’ to the UI to detect the clicking of the **SAVE** button and activate the method **onSave** whenever the **SAVE** button is clicked. Initially, the system will not be able to recognize the **onSave** and flag it as an error. Once the **onSave** method is complete, the error will disappear.

```
buttonSave.setOnClickListener(onSave);
```

If the UI has another button, then there will be another set of these 3 statements to handle the clicking of this 2<sup>nd</sup> button in a similar way.

Equally important, it is the code inside the **onSave** method. The code should read in data entered by the user in the 3 *EditText* fields as well as the selection made by the user in the RadioGroup. See next step. When the following declaration are first typed in, the Android Studio cannot recognize the classes:

```
private EditText restaurantName, restaurantAddress, restaurantTel;  

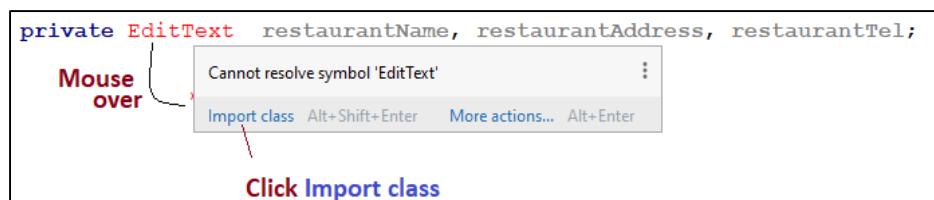
private Button buttonSave;  

private RadioGroup restaurantTypes;
```

???

The IDE is very intelligent. Just do the following. The IDE will import the necessary library.

Repeat for all classes.



14. Update the *RestaurantList* Activity class with the following content.

```
1 package com.sp.restaurantlist;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.Button;
8 import android.widget.EditText;
9 import android.widget.RadioGroup;
10 import android.widget.Toast;
11
12 public class RestaurantList extends AppCompatActivity {
13     private EditText restaurantName;
14     private RadioGroup restaurantTypes;
15     private Button buttonSave;
16     private EditText restaurantAddress;
17     private EditText restaurantTel;
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.main);
23
24         restaurantName = findViewById(R.id.restaurant_name);
25         restaurantTypes = findViewById(R.id.restaurant_types);
26
27         buttonSave = findViewById(R.id.button_save);
28         buttonSave.setOnClickListener(onSave);
29
30         restaurantAddress = findViewById(R.id.restaurant_address);
31         restaurantTel = findViewById(R.id.restaurant_tel);
32     }
33
34     private View.OnClickListener onSave = new View.OnClickListener() {
35         @Override
36         public void onClick(View v) {
37             // To read data from restaurantName EditText
38             String nameStr = restaurantName.getText().toString();
39             // To read data from restaurantAddress EditText
40             String addressStr = restaurantAddress.getText().toString();
41             // To read data from restaurantTel EditText
42             String telStr = restaurantTel.getText().toString();
43             String restType = "";
44             //To read selection of restaurantTypes RadioGroup
45             int radioID = restaurantTypes.getCheckedRadioButtonId();
46             if (radioID == R.id.chinese) {
47                 restType = "Chinese";
48             } else
49             if (radioID == R.id.western) {
50                 restType = "Western";
51             } else
52             if (radioID == R.id.indian) {
53                 restType = "Indian";
54             } else
55             if (radioID == R.id.indonesian) {
56                 restType = "Indonesian";
57             } else
58             if (radioID == R.id.korean) {
59                 restType = "Korean";
60             } else
61             if (radioID == R.id.japanese) {
62                 restType = "Japanese";
63             } else
64             if (radioID == R.id.thai) {
65                 restType = "Thai";
66             }
67             String combineStr = nameStr + "\n" + restType;
68             Toast.makeText(getApplicationContext(), combineStr, Toast.LENGTH_LONG).show();
69         }
70     };
71 }
```

## Notes on String in Java

The following are valid String assignments:

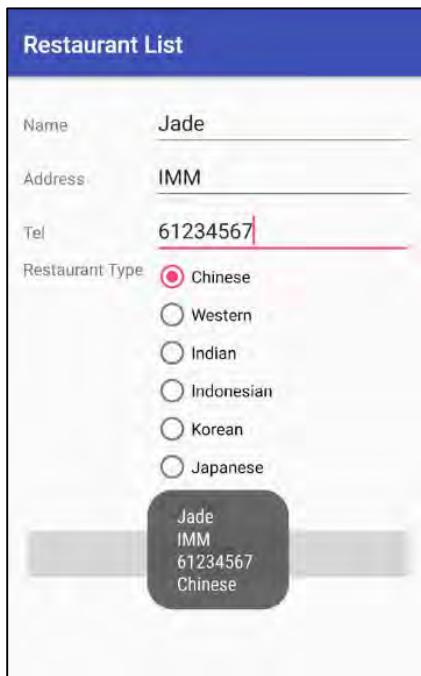
```
String a = "I am here";
String b = "today";
String c = a + " " + b;    // I am here today
String d = a + "\n" + b;  // I am here
                           // today
```

## Further Enhancements

15. Complete the *RestaurantList* Activity class to include reading data from the address and telephone *EditText* widgets and display the result using **Toast**. Use the variables `restaurantAddress` and `restaurantTel` for the address and telephone *EditText* widgets respectively.

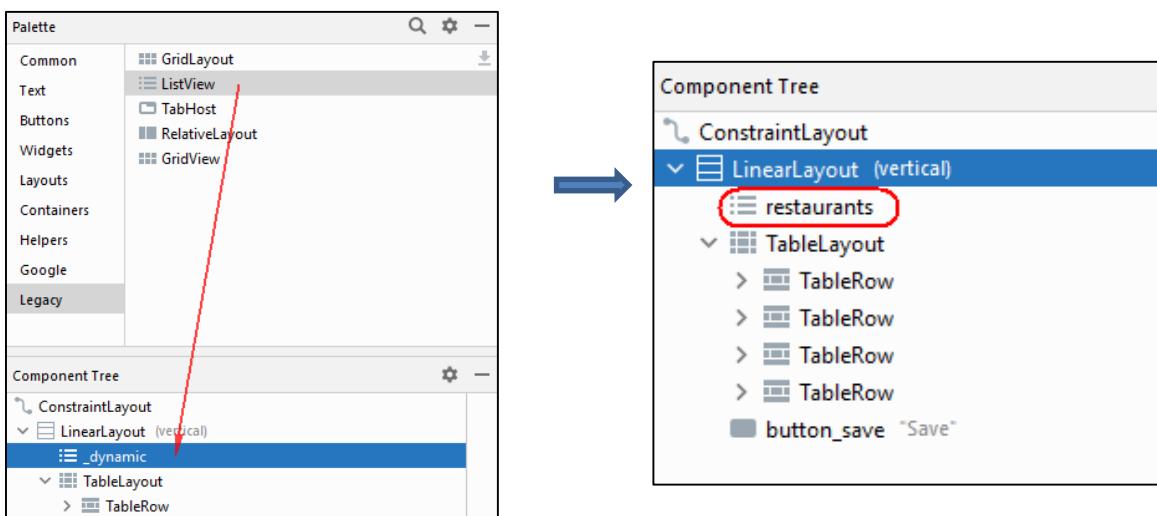
A **toast** provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

### 16. Show your lecturer after you have achieved the Further Enhancements.



## Part V – Creating a Form with List and Model Class to Hold Restaurant Data

17. In this part of the exercise, List and Model class will be added
18. Create a new project *Lab1b* with **No Activity** and the following information:
  - **Name:** *Restaurant List*
  - **Package name:** *com.sp.restaurantlist*
  - **Save location:** *C:\MAD\AndroidStudioProjects\Lab1b or D:\MAD\AndroidStudioProjects\Lab1b*
  - **Language:** *Java*
  - **Minimum SDK:** *API 27: Android 8.1 (Oreo)*
  - **Source Language:** *Java*
  - **Build configuration language:** *Groovy DSL (build.gradle)*
19. Open Windows File Explorer and navigate to your Android Studio workspace (*C:\MAD\AndroidStudioProjects* or *D:\MAD\AndroidStudioProjects*) where all your projects are created and saved
20. Double click folder name *Lab1a* to open the *Lab1a* project folder and navigate down to “*app\src\main*” folder. Copy **AndroidManifest.xml** file, **java** and **res** folders
21. Go to newly created project “*Lab1b\app\src\main*” folder and paste into it to overwrite existing files and folders
22. Go back to Android Studio (project *Lab1b*). Open the *RestaurantList.java* and *main.xml* files and they should show the exact same content from *Lab1a*
23. To enhance the *Restaurant List* app, a *ListView* widget will be added to the UI View to display a list of the restaurants information entered
24. Open the *main.xml* at Design pane. Drag a *ListView* widget from **Legacy** option and drop into the **LinearLayout**.

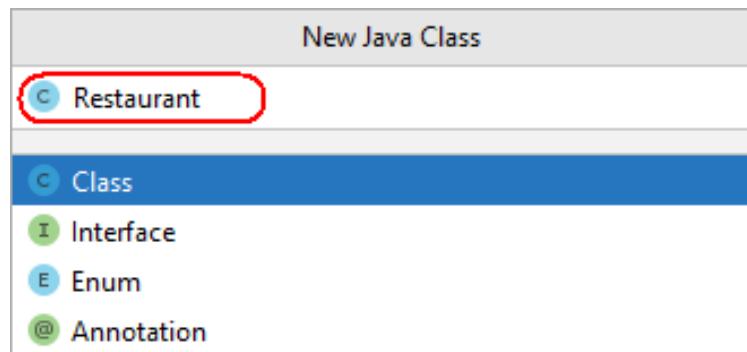
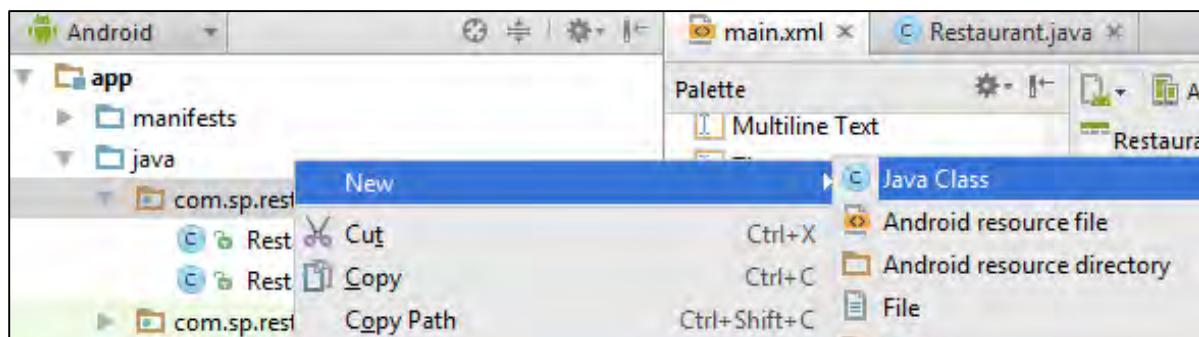


25. Update the *ListView ID* to “**restaurants**” and set the *layout\_height* to **80dp**.



26. With the *ListView* widget created, every new set of restaurant information will be treated as a record (which contains *restaurantName*, *restaurantAddress*, *restaurantTel* and *restaurantTypes* widgets data)
27. Individual restaurant's record will be stored using **Restaurant Data Model** and added to **ArrayList**
28. Data entered in the form is collected and kept in memory using **ArrayList**
29. To display all the records collected in *ArrayList* to *ListView* on the form, an **ArrayAdapter** (acts as Controller) will update the *ListView* whenever a new restaurant record is added to the **ArrayAdapter** (update both **ArrayList** and *ListView* automatically)
30. To create the Restaurant Data Model class, right click on **com.sp.restaurantlist** folder and select **New > Java Class**

31. Enter the **Java Class** name as **Restaurant** and double click “**Class**”.



32. Complete the *Restaurant.java* class with the following code and save

```
1 package com.sp.restaurantlist;
2
3 public class Restaurant {
4     private String restaurantName = "";
5     private String restaurantAddress = "";
6     private String restaurantTel = "";
7     private String restaurantType = "";
8
9     public String getName() { return restaurantName; }
10
11    public void setName(String restaurantName) { this.restaurantName = restaurantName; }
12
13    public String getAddress() { return restaurantAddress; }
14    public void setAddress(String restaurantAddress) { this.restaurantAddress = restaurantAddress; }
15
16    public String getTelephone() { return restaurantTel; }
17
18    public void setTelephone(String restaurantTel) { this.restaurantTel = restaurantTel; }
19
20    public String getRestaurantType() { return restaurantType; }
21
22    public void setRestaurantType(String restaurantType) { this.restaurantType = restaurantType; }
23
24    @Override
25    public String toString() { return (getName()); }
26
27 }
```

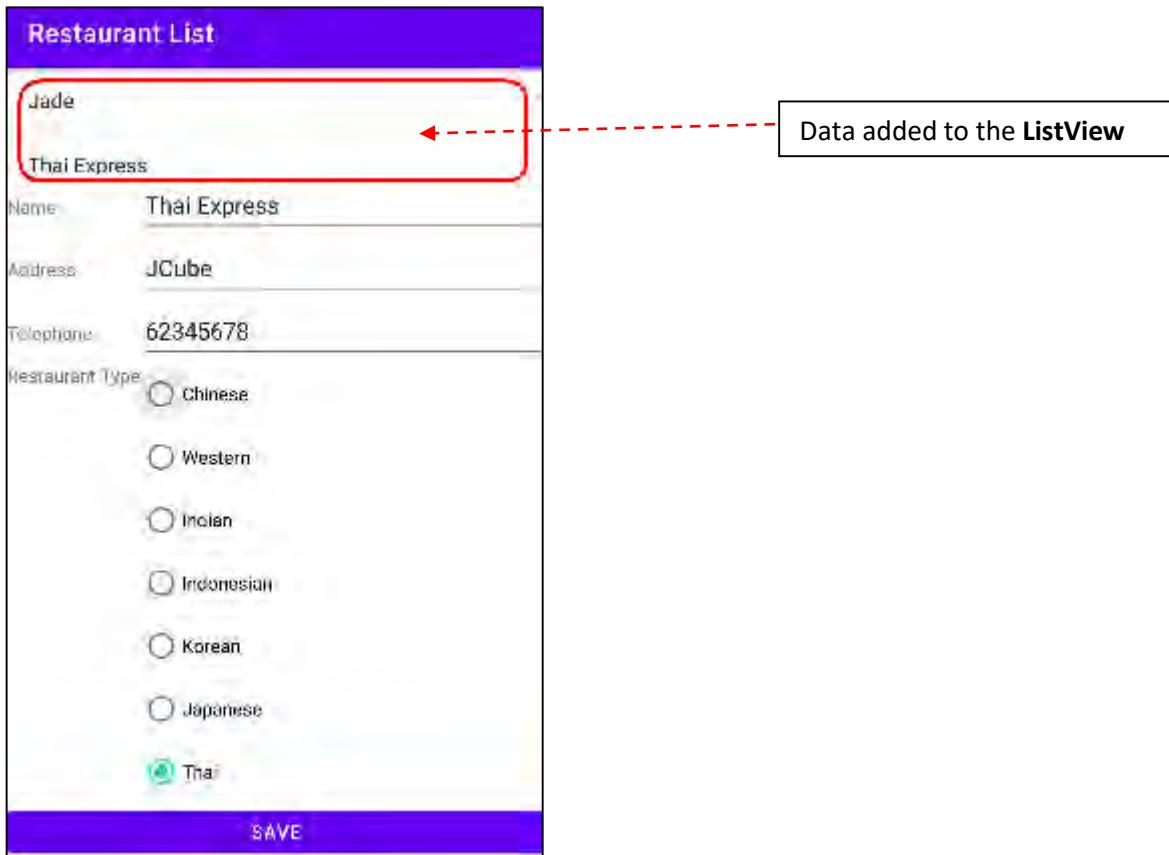
33. Create the Controller (ArrayAdapter object), Data Model (ArrayList object with Restaurant type element), and View (ListView object). Link up Data Model with View via Controller in the *RestaurantList* Activity class and save.

```
1 package com.sp.restaurantlist;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.ArrayAdapter;
8 import android.widget.Button;
9 import android.widget.EditText;
10 import android.widget.ListView;
11 import android.widget.RadioGroup;
12 import android.widget.Toast;
13
14 import java.util.ArrayList;
15 import java.util.List;
16
17 public class RestaurantList extends AppCompatActivity {
18     private EditText restaurantName;
19     private RadioGroup restaurantTypes;
20     private Button buttonSave;
21     private EditText restaurantAddress;
22     private EditText restaurantTel;
23
24     private List<Restaurant> model = new ArrayList<Restaurant>();
25     private ArrayAdapter<Restaurant> adapter = null;
26     private ListView list;
```

Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

```
28     @Override
29     protected void onCreate(Bundle savedInstanceState) {
30         super.onCreate(savedInstanceState);
31         setContentView(R.layout.main);
32
33         restaurantName = findViewById(R.id.restaurant_name);
34         restaurantTypes = findViewById(R.id.restaurant_types);
35
36         buttonSave = findViewById(R.id.button_save);
37         buttonSave.setOnClickListener(onSave);
38         restaurantAddress = findViewById(R.id.restaurant_address);
39         restaurantTel = findViewById(R.id.restaurant_tel);
40
41         list = findViewById(R.id.restaurants);
42         adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, model);
43         list.setAdapter(adapter);
44     }
45
46     private View.OnClickListener onSave = new View.OnClickListener() {
47         @Override
48         public void onClick(View v) {
49             // To read data from restaurantName EditText
50             String nameStr = restaurantName.getText().toString();
51             // To read data from restaurantAddress EditText
52             String addressStr = restaurantAddress.getText().toString();
53             // To read data from restaurantTel EditText
54             String telStr = restaurantTel.getText().toString();
55             String restType = "";
56             //To read selection of restaurantTypes RadioGroup
57             int radioID = restaurantTypes.getCheckedRadioButtonId();
58             if (radioID == R.id.chinese) {
59                 restType = "Chinese";
60             } else
61             if (radioID == R.id.western) {
62                 restType = "Western";
63             } else
64             if (radioID == R.id.indian) {
65                 restType = "Indian";
66             } else
67             if (radioID == R.id.indonesian) {
68                 restType = "Indonesian";
69             } else
70             if (radioID == R.id.korean) {
71                 restType = "Korean";
72             } else
73             if (radioID == R.id.japanese) {
74                 restType = "Japanese";
75             } else
76             if (radioID == R.id.thai) {
77                 restType = "Thai";
78             }
79
80             String combineStr = nameStr + "\n" + addressStr + "\n" + telStr + "\n" + restType;
81             Toast.makeText(v.getContext(), combineStr, Toast.LENGTH_LONG).show();
82             //Create Restaurant Data Model
83             Restaurant restaurant = new Restaurant();
84             //Update the Restaurant Data Model
85             restaurant.setName(nameStr);
86             restaurant.setAddress(addressStr);
87             restaurant.setTelephone(telStr);
88             restaurant.setRestaurantType(restType);
89             //Pass the record to ArrayAdapter.
90             //It will update the ListArray and the ListView
91             adapter.add(restaurant);
92
93     };
94 }
95 }
```

34. Run the *Lab1b* project. The **ListView** will remain empty until we do something to populate it
35. Enter some test data and click the **Save** button



36. **Show to your lecturer after you have completed the exercise.**

## Part VI – BMI Calculator (Optional)

1. Develop a BMI calculator app that prompts the user for the weight in **kilogram** and height in **meter**. The formula to compute BMI is as follows:

$$\text{BMI} = \text{weight} / (\text{height} * \text{height})$$

2. A sample run of the app is as shown:



The screenshot shows a mobile application titled "BMI Calculator". It has two input fields: "Weight (kg)" containing "72" and "Height (m)" containing "1.74". Below these is a "CALCULATE" button. At the bottom, the result is displayed as "BMI - 23.78 Healthy".

When a user clicks on the "**Calculate**" button, the app will get the values entered by the user for the weight and height, then compute and display the BMI value.

**Note: format the BMI value to 2 decimal places and ensure that the weight and height inputs only accept numeric values.**

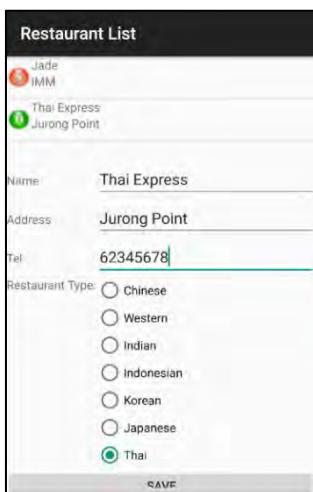
3. With the BMI value, display the result on the user interface together with the diagnostic associated to the BMI value using the following table:

| BMI            | Weight status |
|----------------|---------------|
| Below 18.5     | Underweight   |
| 18.5–24.9      | Healthy       |
| 25.0–29.9      | Overweight    |
| 30.0 and above | Obese         |

-END-

## Practical 2: List and Tab View

In this session, you will learn to create a List View for the restaurant list with a basic customization and to create a Tab View to split the restaurant list and detail form.



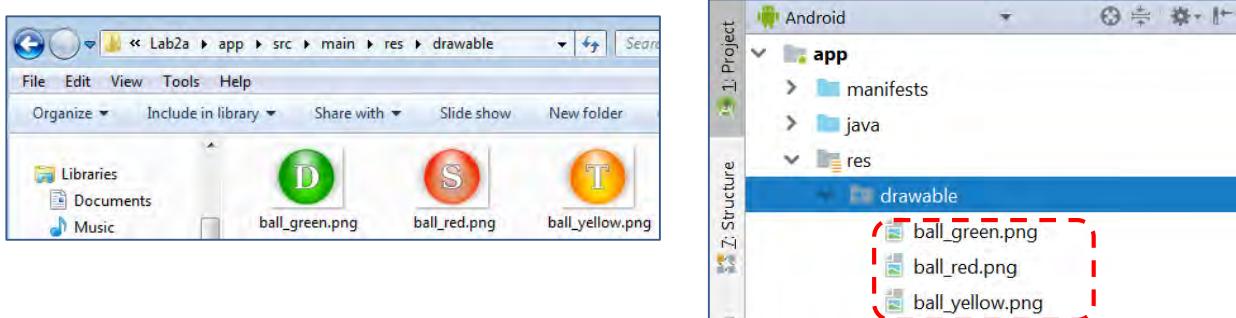
### Part I – Creating a Fancier List

1. Create a new project with **No Activity** and the following information:
  - **Name:** *Restaurant List*
  - **Package name:** *com.sp.restaurantlist*
  - **Save location:** *C:\MAD\AndroidStudioProjects\Lab2a or D:\MAD\AndroidStudioProjects\Lab2a*
  - **Language:** *Java*
  - **Minimum SDK:** *API 27: Android 8.1 (Oreo)*
  - **Source Language:** *Java*
  - **Build configuration language:** *Groovy DSL (build.gradle)*
2. Open **Windows File Explorer** and navigate to your Android Studio workspace (*C:\MAD\AndroidStudioProjects* or *D:\MAD\AndroidStudioProjects*) where all your projects are created and saved.
3. Double click to open the *Lab1b* project folder and navigate down to “**app\src\main**” folder. Copy **AndroidManifest.xml** file, **java** and **res** folders
4. Go to newly created project “*Lab2a\app\src\main*” folder and paste into it to overwrite existing folders and files
5. Go back to Android Studio. Open the *RestaurantList.java* and *main.xml* files and they should show the content from *Lab1b* (which is currently a copy of *Lab1b*)
6. To improve the UI View of **List View** created in *Lab1b*, a *row.xml* layout file will be created. Instead of displaying the restaurant's name only (which is a simple String) for each restaurant added to the list, the new row view will contain an image icon, restaurant name and address (example shown below)

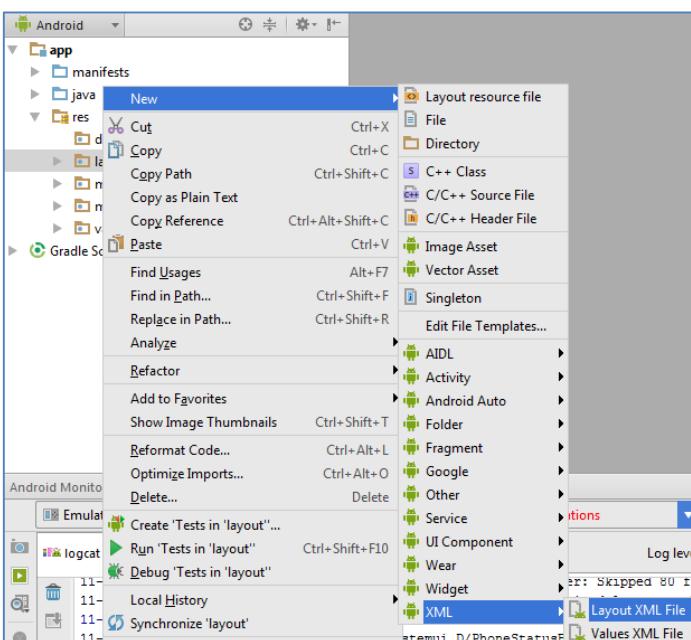


Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

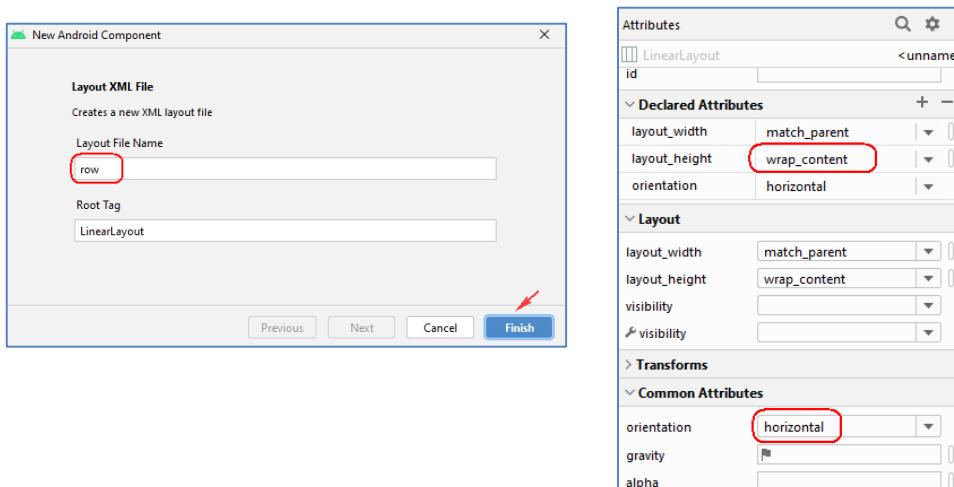
- Before working on the customized view, download the 3 image files (*ball\_red.png*, *ball\_yellow.png*, and *ball\_green.png*) from Brightspace under **Practical Session Resources > “Image icons might be used in Labs”** folder. Right click on the image and select ‘**save link as**’ to save the image into **Lab2a\app\src\main\res\drawable** folder.



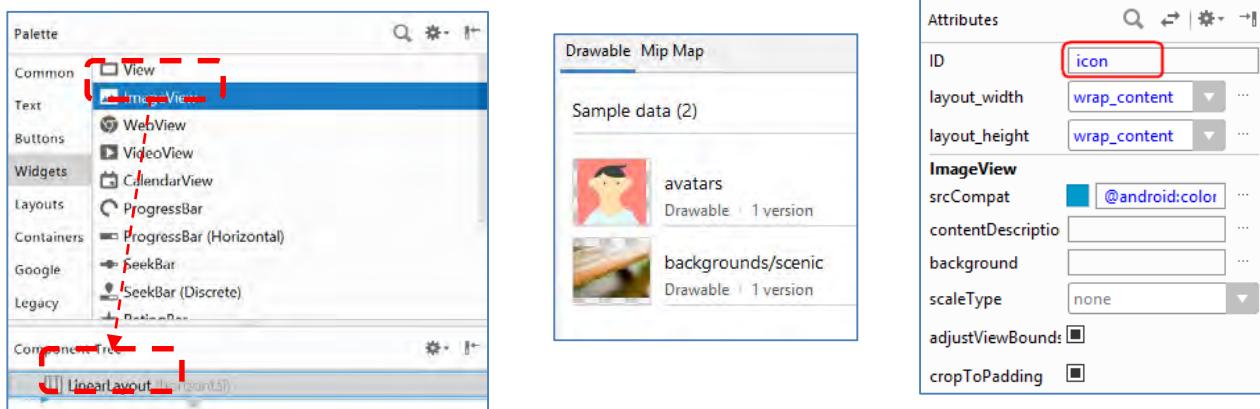
- To achieve the customize row view, we need a layout file just for this with a nested pair of **LinearLayout** containers. Right click on the **res/layout** folder and select **New > XML > Layout XML File**. Enter the layout file name as '**row**'.



- Set the **LinearLayout layout\_height** to **wrap\_content** and **orientation** attribute to **horizontal**



10. Drag-and-drop an **ImageView** widget from **Widgets** option into the **Horizontal LinearLayout**. Choose any icon in the **Drawable** tab and change the **ID** to '**icon**'



If the following error is encountered:

**error: '@tools:sample/avatars' is incompatible with attribute src (attr) reference|color.**

Open the **row.xml** in Code tab and remove the line "**android:src="@tools:sample/avatars"**:

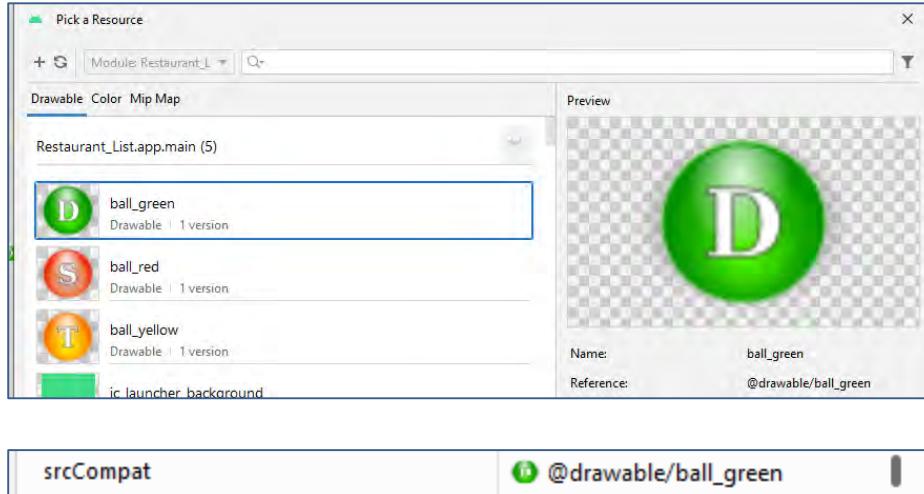
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:src="@tools:sample/avatars" />           remove this line
</LinearLayout>
```

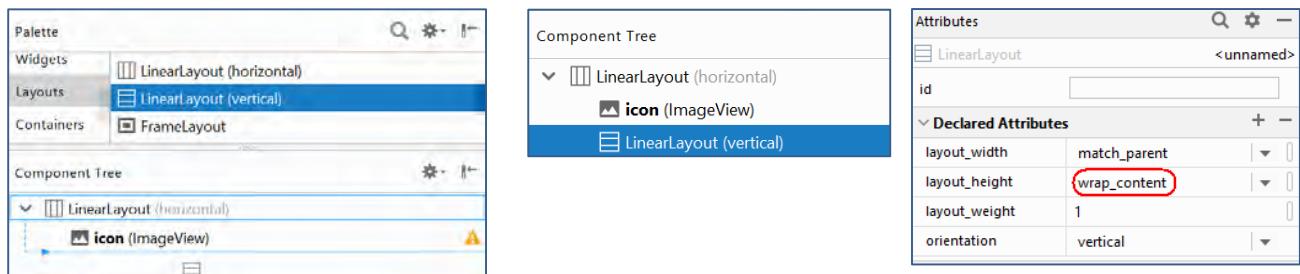
11. Select **imageView**. Click on the for **srcCompat**.



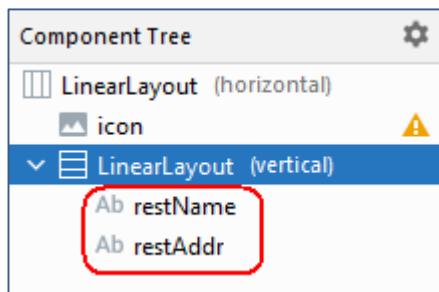
Select **ball\_green**.



12. Drag-and-drop a **LinearLayout (vertical)** from **Layouts** option into the current **LinearLayout (horizontal)** below **ImageView** widget and set the layout height to **wrap\_content**



13. Drag-and-drop two **TextView** widgets into the **Vertical LinearLayout**. Change the IDs of the two **TextView** widgets to '*restName*' and '*restAddr*'



14. In order to make use of the new customized *row.xml* layout to display *Restaurant* Data Model in the **ListView**, a subclass of  **ArrayAdapter** will be created to link the *row.xml* layout with the Data Model

```
class RestaurantAdapter extends ArrayAdapter<Restaurant> {
    RestaurantAdapter() {
        super(RestaurantList.this, R.layout.row, model);
    }
}
```

15. Open *RestaurantList.java* file and update the content:

```
1  package com.sp.restaurantlist;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.view.LayoutInflater;
6  import android.view.View;
7  import android.view.ViewGroup;
8  import android.widget.ArrayAdapter;
9  import android.widget.Button;
10 import android.widget.EditText;
11 import android.widget.ImageView;
12 import android.widget.ListView;
13 import android.widget.RadioGroup;
14 import android.widget.TextView;
15
16 import java.util.ArrayList;
17 import java.util.List;
18
19 public class RestaurantList extends AppCompatActivity {
20     private EditText restaurantName;
21     private RadioGroup restaurantTypes;
22     private Button buttonSave;
23     private EditText restaurantAddress;
24     private EditText restaurantTel;
25
26     private List<Restaurant> model = new ArrayList<>();
27     private RestaurantAdapter adapter = null;
28     private ListView list;
29
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.main);
34
35         restaurantName = findViewById(R.id.restaurant_name);
36         restaurantTypes = findViewById(R.id.restaurant_types);
37
38         buttonSave = findViewById(R.id.button_save);
39         buttonSave.setOnClickListener(onSave);
40
41         restaurantAddress = findViewById(R.id.restaurant_address);
42         restaurantTel = findViewById(R.id.restaurant_tel);
43
44         list = findViewById(R.id.restaurants);
45         adapter = new RestaurantAdapter();
46         list.setAdapter(adapter);
47     }
}
```

```
48
49
50
51  ↗
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
```

```
private View.OnClickListener onSave = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // To read data from restaurantName EditText
        String nameStr = restaurantName.getText().toString();
        // To read data from restaurantAddress EditText
        String addressStr = restaurantAddress.getText().toString();
        // To read data from restaurantTel EditText
        String telStr = restaurantTel.getText().toString();
        String restType = "";
        // To read data from restaurantTel EditText
        String telStr = restaurantTel.getText().toString();
        String restType = "";
        //To read selection of restaurantTypes RadioGroup
        int radioID = restaurantTypes.getCheckedRadioButtonId();
        if (radioID == R.id.chinese) {
            restType = "Chinese";
        } else
        if (radioID == R.id.western) {
            restType = "Western";
        } else
        if (radioID == R.id.indian) {
            restType = "Indian";
        } else
        if (radioID == R.id.indonesian) {
            restType = "Indonesian";
        } else
        if (radioID == R.id.korean) {
            restType = "Korean";
        } else
        if (radioID == R.id.japanese) {
            restType = "Japanese";
        } else
        if (radioID == R.id.thai) {
            restType = "Thai";
        }
        //String combineStr = nameStr + "\n" + addressStr + "\n" + telStr + "\n" +restType;
        //Toast.makeText(v.getContext(), combineStr, Toast.LENGTH_LONG).show();
        Restaurant restaurant = new Restaurant();
        restaurant.setName(nameStr);
        restaurant.setAddress(addressStr);
        restaurant.setTelephone(telStr);
        restaurant.setRestaurantType(restType);
        adapter.add(restaurant);
    }
};
```

```
94
95
96
97
98
99 @
100 static class RestaurantHolder {
101     private TextView restName = null;
102     private TextView addr = null;
103     private ImageView icon = null;
104     RestaurantHolder(View row) {
105         restName = row.findViewById(R.id.restName);
106         addr = row.findViewById(R.id.restAddr);
107         icon = row.findViewById(R.id.icon);
108     }
109     void populateFrom(Restaurant r) {
110         restName.setText(r.getName());
111         addr.setText(r.getAddress());
112         if (r.getRestaurantType().equals("Chinese")) {
113             icon.setImageResource(R.drawable.ball_red);
114         } else if (r.getRestaurantType().equals("Western")) {
115             icon.setImageResource(R.drawable.ball_yellow);
116         } else {
117             icon.setImageResource(R.drawable.ball_green);
118         }
119     }
120 }
121
122
123 @
124 class RestaurantAdapter extends ArrayAdapter<Restaurant> {
125     RestaurantAdapter() { super(RestaurantList.this,R.layout.row, model); }
126
127     @Override
128     public View getView(int position, View convertView, ViewGroup parent) {
129         View row = convertView;
130         RestaurantHolder holder;
131         if (row == null) {
132             LayoutInflater inflater = getLayoutInflater();
133             row = inflater.inflate(R.layout.row, parent, false);
134             holder = new RestaurantHolder(row);
135             row.setTag(holder);
136         } else {
137             holder = (RestaurantHolder)row.getTag();
138         }
139         holder.populateFrom(model.get(position));
140         return (row);
141     }
142 }
143 }
```

16. Click on the  icon at the top menu bar to run the *Lab2a* project. If successful, an emulator will be launched with the project app running. Try entering one or two record and save. You will see the rows are displayed with the new customized format.

RestaurantAdapter and RestaurantHolder are difficult concepts especially for first timers.

Besides the surface code, the understanding of how Android Studio handles view display in ListView is crucial. Try to use as many LogCat as possible to fully trace out the behaviors of the code in this section.

**Restaurant List**

|                           |                                     |
|---------------------------|-------------------------------------|
| Jade IMM                  | <input type="checkbox"/>            |
| Thai Express Jurong Point | <input checked="" type="checkbox"/> |

Name

Address

Tel

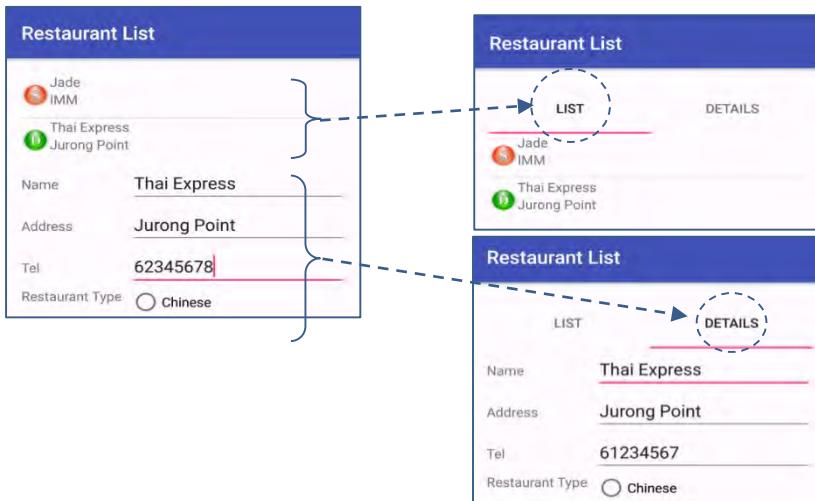
Restaurant Type:

- Chinese
- Western
- Indian
- Indonesian
- Korean
- Japanese
- Thai

**SAVE**

## Part II – Creating Tab View

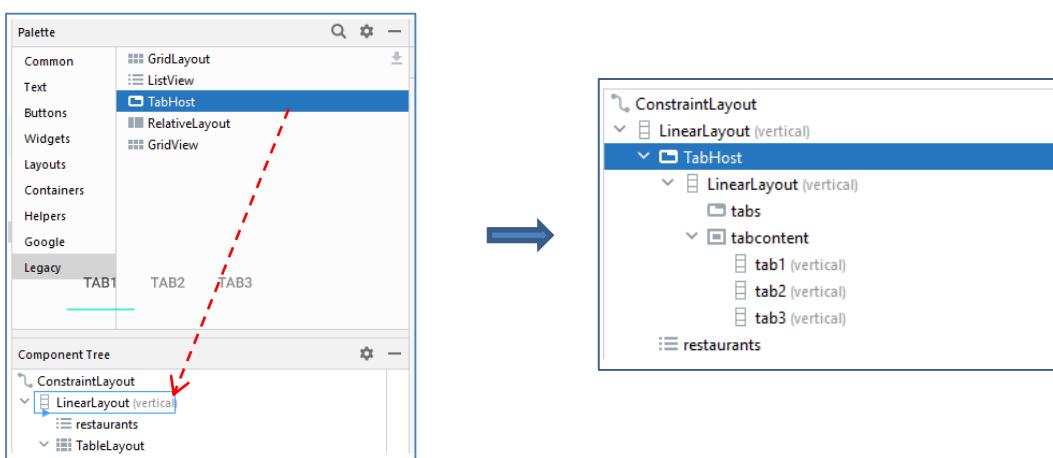
17. In this lab, we will separate our **ListView** and data entry form into two views using tabs. A **TabHost** widget and **tabContent** will be used to implement the new layout.
18. There are 2 tabs: *List* and *Details*. The *Details* contains the input form and a SAVE button. Whenever a new restaurant record is saved at *Details* tab, a new entry will be added to the restaurant list at *List* tab



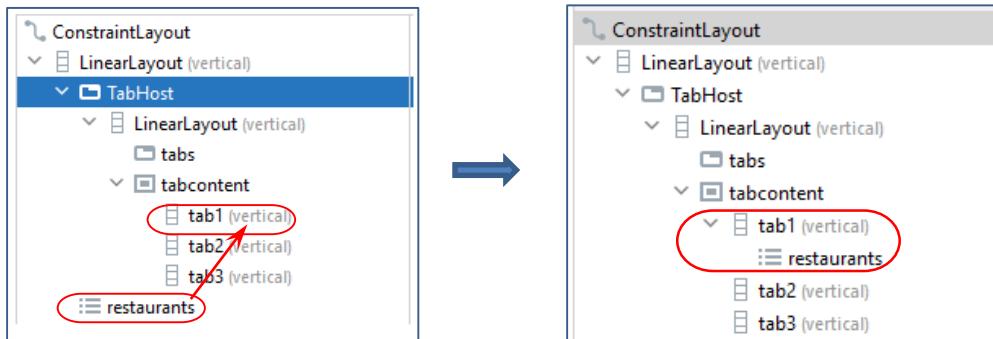
19. Create a new project with **No Activity** and the following information:

- **Name:** Restaurant List
- **Package name:** com.sp.restaurantlist
- **Save location:** C:\MAD\AndroidStudioProjects\Lab2b or D:\MAD\AndroidStudioProjects\Lab2b
- **Language:** Java
- **Minimum SDK:** API 27: Android 8.1 (Oreo)
- **Source Language:** Java
- **Build configuration language:** Groovy DSL (build.gradle)

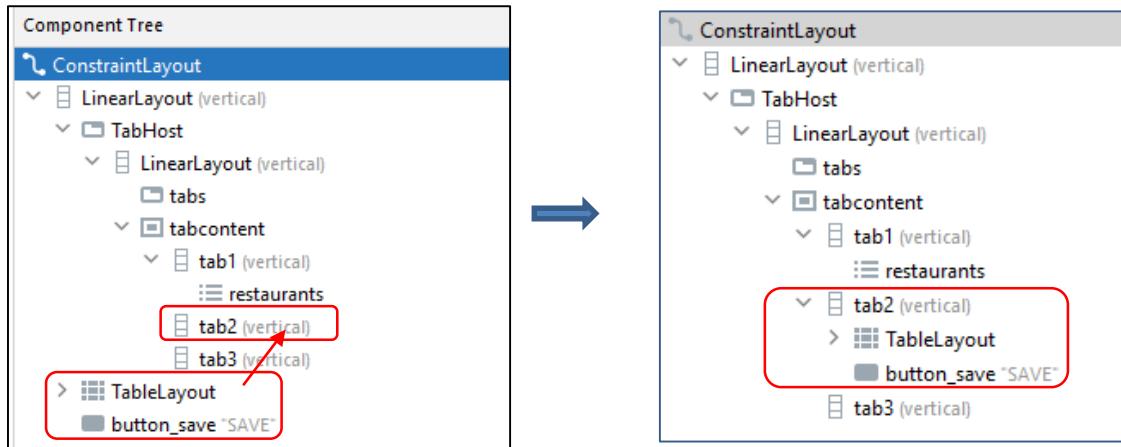
20. Open your **Windows File Explorer** and navigate to your **Android Studio** workspace where all your projects are created.
21. Copy **AndroidManifest.xml** file, **java** and **res** folders from **Lab2a\app\src\main** project folder and paste into **Lab2b\app\src\main** folder to overwrite the existing files and folders
22. At Android Studio, *Lab2b* project content will contain the newly copied files and folders
23. To create a tab View as shown below, open the *main.xml* file. Drag a **TabHost** widget from **Legacy** option in Palette and drop it to the **LinearLayout** as child tag. Update **TabHost** widget ID to **tabHost**. Set **layout\_width** to match\_parent and **layout\_height** to wrap\_content



24. Drag **restaurants ListView** widget into **tab1** as child tag



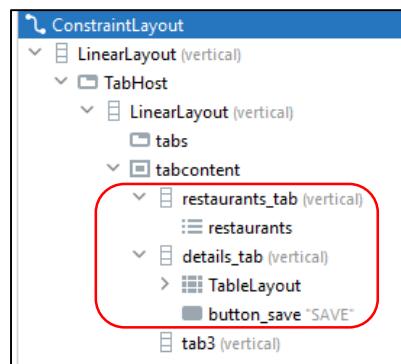
25. Drag both **TableLayout** and **button\_save Button** widgets and drop into **tab2** as a child tag



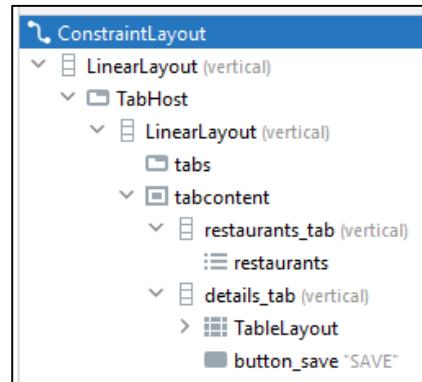
26. Update the IDs for the tab1 and tab2.

**tab1** – ID: *restaurants\_tab*

**tab2** – ID: *details\_tab*



27. Delete **tab3** widget.



28. With the new layout created, we will learn how to create tabs and load the respective View to each tab
29. Every tab has a tab indicator and a tag that is used to keep track of it (the *host* variable below is referencing the *TabHost* widget with id of *tabHost*)

```
host = findViewById(R.id.tabHost);  
  
//Tab 1  
TabHost.TabSpec spec = host.newTabSpec("List");  
spec.setContent(R.id.restaurants_tab);  
spec.setIndicator("List");  
host.addTab(spec);  
  
//Tab 2  
spec = host.newTabSpec("Details");  
spec.setContent(R.id.details_tab);  
spec.setIndicator("Details");  
host.addTab(spec);
```

30. **TabSpec** is used to set Indicator, Content, Label, Icon, etc. on the particular tab. Two *TabSpec* objects created are initialized by ***newTabSpec*** method of TabHost class, which will have tag or title as "List" and "Details".
31. ***setContent*** will load the respective tab with the view contains by the *restaurants\_tab* and *details\_tab*.
32. ***setIndicator*** will indicate TabHost that particular tab is selected or not
33. ***setCurrentTab*** will activate the current tab view to the tab number specified. The first tab ('List' for *List* tab) created will be having an index number 0

```
host.setCurrentTab(0);
```

34. Open the *RestaurantList.java* file, update the content and save

You may encounter 'deprecated' notice when updating the code for the TabHost.

Ignore and continue typing.

You can still use TabHost.

It just means Android Studio has other better means of doing this (via fragments).

```
host = findViewById(R.id.tabHost);  
host.setup();  
  
TabHost.TabSpec spec=host.newTabSpec( tag: "List");  
spec.setContent(R.id.restaurants_tab);  
spec.setIndicator("List");  
host.addTab(spec);  
spec=host.newTabSpec( tag: "Details");  
spec.setContent(R.id.details_tab);  
spec.setIndicator("Details");  
host.addTab(spec);  
host.setCurrentTab(0);
```

```
1 package com.sp.restaurantlist;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.LayoutInflater;
6 import android.view.View;
7 import android.view.ViewGroup;
8 import android.widget.AdapterView;
9 import android.widget.ArrayAdapter;
10 import android.widget.Button;
11 import android.widget.EditText;
12 import android.widget.ImageView;
13 import android.widget.ListView;
14 import android.widget.RadioGroup;
15 import android.widget.TabHost;
16 import android.widget.TextView;
17
18 import java.util.ArrayList;
19 import java.util.List;
20
21 public class RestaurantList extends AppCompatActivity {
22     private EditText restaurantName;
23     private RadioGroup restaurantTypes;
24     private Button buttonSave;
25     private EditText restaurantAddress;
26     private EditText restaurantTel;
27
28     private List<Restaurant> model = new ArrayList<>();
29     private RestaurantAdapter adapter = null;
30
31     private ListView list;
32     private TabHost host;
33
34     @Override
35     protected void onCreate(Bundle savedInstanceState) {
36         super.onCreate(savedInstanceState);
37         setContentView(R.layout.main);
38
39         restaurantName = findViewById(R.id.restaurant_name);
40         restaurantTypes = findViewById(R.id.restaurant_types);
41
42         buttonSave = findViewById(R.id.button_save);
43         buttonSave.setOnClickListener(onSave);
44
45         restaurantAddress = findViewById(R.id.restaurant_address);
46         restaurantTel = findViewById(R.id.restaurant_tel);
47
48         list = findViewById(R.id.restaurants);
49         adapter = new RestaurantAdapter();
50         list.setAdapter(adapter);
51
52         host = findViewById(R.id.tabHost);
53         host.setup();
54
55         //Tab 1
56         TabHost.TabSpec spec = host.newTabSpec("List");
57         spec.setContent(R.id.restaurants_tab);
58         spec.setIndicator("List");
59         host.addTab(spec);
```

```
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

    //Tab 2
    spec = host.newTabSpec("Details");
    spec.setContent(R.id.details_tab);
    spec.setIndicator("Details");
    host.addTab(spec);
    host.setCurrentTab(0);

}

private View.OnClickListener onSave = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // To read data from restaurantName EditText
        String nameStr = restaurantName.getText().toString();
        // To read data from restaurantAddress EditText
        String addressStr = restaurantAddress.getText().toString();
        // To read data from restaurantTel EditText
        String telStr = restaurantTel.getText().toString();

        String restType = "";
        //To read selection of restaurantTypes RadioGroup
        int radioID = restaurantTypes.getCheckedRadioButtonId();
        if (radioID == R.id.chinese ) {
            restType = "Chinese";
        } else
        if (radioID == R.id.western ) {
            restType = "Western";
        } else

        if (radioID == R.id.indian ) {
            restType = "Indian";
        } else
        if (radioID == R.id.indonesian ) {
            restType = "Indonesian";
        } else
        if (radioID == R.id.korean) {
            restType = "Korean";
        } else
        if (radioID == R.id.japanese) {
            restType = "Japanese";
        } else
        if (radioID == R.id.thai) {
            restType = "Thai";
        }

        //String combineStr = nameStr + "\n" + addressStr + "\n" + telStr + "\n" +restType;
        //Toast.makeText(v.getContext(), combineStr, Toast.LENGTH_LONG).show();
        Restaurant restaurant = new Restaurant();
        restaurant.setName(nameStr);
        restaurant.setAddress(addressStr);
        restaurant.setTelephone(telStr);
        restaurant.setRestaurantType(restType);

        adapter.add(restaurant);
    }
};
```

```
115
116
117
118
119
120 static class RestaurantHolder {
121     private TextView restName = null;
122     private TextView addr = null;
123     private ImageView icon = null;
124     RestaurantHolder(View row) {
125         restName = row.findViewById(R.id.restName);
126         addr = row.findViewById(R.id.restAddr);
127         icon = row.findViewById(R.id.icon);
128     }
129     void populateFrom(Restaurant r) {
130         restName.setText(r.getName());
131         addr.setText(r.getAddress());
132         if (r.getRestaurantType().equals("Chinese")) {
133             icon.setImageResource(R.drawable.ball_red);
134         } else if (r.getRestaurantType().equals("Western")) {
135             icon.setImageResource(R.drawable.ball_yellow);
136         } else {
137             icon.setImageResource(R.drawable.ball_green);
138         }
139     }
140
141     class RestaurantAdapter extends ArrayAdapter<Restaurant> {
142         RestaurantAdapter() { super(RestaurantList.this,R.layout.row, model); }
143
144         @Override
145         public View getView(int position, View convertView, ViewGroup parent) {
146             View row = convertView;
147             RestaurantHolder holder;
148             if (row == null) {
149                 LayoutInflator inflater = getLayoutInflater();
150                 row = inflater.inflate(R.layout.row, parent, false);
151                 holder = new RestaurantHolder(row);
152                 row.setTag(holder);
153             } else {
154                 holder = (RestaurantHolder)row.getTag();
155             }
156             holder.populateFrom(model.get(position));
157             return (row);
158         }
159     }
160 }
```

35. Click on the  icon to run *Lab2b* project. If successful, an emulator will be launched and display the **List** tab.  
Try entering two new restaurant records and save. Click on the **List** tab, you will see two rows have been added to the **ListView**



36. Show your lecturer when you have completed this section of exercise

### Part III – Detecting List Click

37. In previous part of the work, we have successfully create tabs to separate Restaurant form and **ListView** with tabs
38. Try to click on any one of the records listed in the List tab. Nothing will happen! This is because to capture the List item click, we need to activate item click listener of **ListView** (same concept as capturing Button click)

```
list.setOnItemClickListener(onListClick);
```

39. When an item from the **ListView** is clicked on, the Controller will pass the event to the **onListClick** which is an **AdapterView** item click listener created to handle the event

```
private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
```

40. Open the *RestaurantList.java* file, **referring to the codes in step 34**, update with the following **highlighted** content and save.

- This is a new method **onListClick()**. Add the **highlighted** codes after line 158 and before line 159. The new codes is as follows:

```
158 }  
159  
160 AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {  
161     @Override  
162     public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
163         Restaurant r = model.get(position);  
164  
165         restaurantName.setText(r.getName());  
166         restaurantAddress.setText(r.getAddress());  
167         restaurantTel.setText(r.getTelephone());  
168  
169         if (r.getRestaurantType().equals("Chinese")) {  
170             restaurantTypes.check(R.id.chinese);  
171         } else if (r.getRestaurantType().equals("Western")) {  
172             restaurantTypes.check(R.id.western);  
173         } else if (r.getRestaurantType().equals("Indian")) {  
174             restaurantTypes.check(R.id.indian);  
175         } else if (r.getRestaurantType().equals("Indonesia")) {  
176             restaurantTypes.check(R.id.indonesian);  
177         } else if (r.getRestaurantType().equals("Korean")) {  
178             restaurantTypes.check(R.id.korean);  
179         } else if (r.getRestaurantType().equals("Japanese")) {  
180             restaurantTypes.check(R.id.japanese);  
181         } else {  
182             restaurantTypes.check(R.id.thai);  
183         }  
184         host.setCurrentTab(1);  
185     }  
186 };
```

- b) Edit the `onCreate()` method. Add the **highlighted** code on line 66, before line 67 to register the list with the listener. The new line of code is as follows:

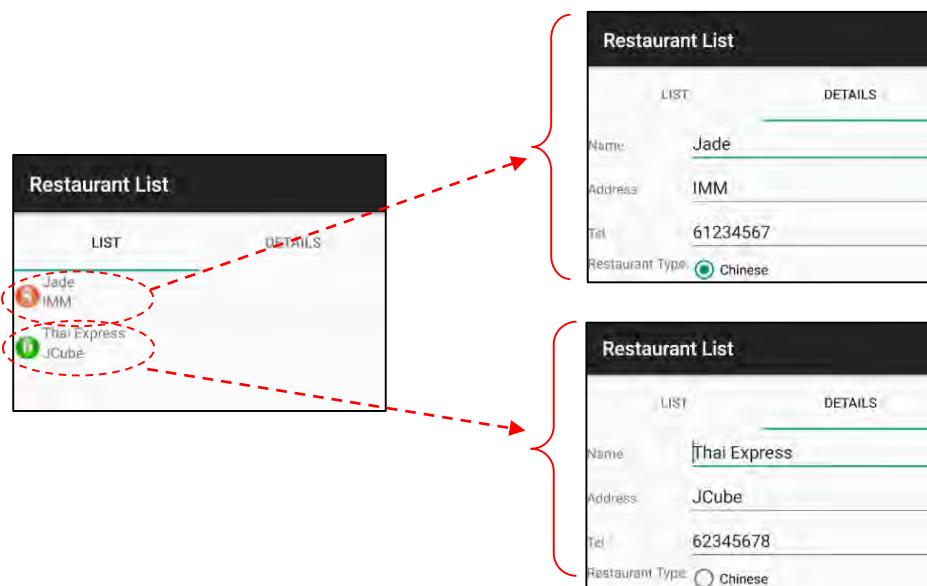
```

64         host.addTab(spec);
65         host.setCurrentTab(0);
66         list.setOnItemClickListener(onListClick);
67     }
  
```

Note that the listener type is not `setOnClickListener` but `setOnItemClickListener`.

Conduct your own experiment to find out the difference if you are curious.

41. Run *Lab2b* project. If successful, an emulator will be launched and display the **List** tab. Enter two new restaurant data and save. Click on the **List** tab, select the first record in the list. The view will switch to **Details** tab and populate with the restaurant data of the record selected



#### 42. Show your lecturer when you have completed this section of exercise

### Further Enhancements

43. Modify the application

- to display the “**DETAILS**” tab by default whenever the app is launched and improve the row of the List display to include **telephone number** as shown (**Hint:** Combine the address and telephone number as string before displaying on view)
- to switch the view to the “**LIST**” tab automatically when the ‘Save’ button is pressed



#### 44. Show your lecturer when you have completed the extra credit

## Part IV – List of EEE Courses (Optional)

- Using **Lab2a** as a reference, design a list view to list all the courses offered by the school of EEE. A sample run of the app is as shown:



- Each list item consists of the course **title**, course **description** and an image.
- Each course is identified by the course **code** e.g. S90, S53 etc.

**Hint: use the course code to determine which course image to display.**

- Like in **Lab2a**, the **"Restaurant" class** store all related data of a restaurant as one record, similarly, you need to create a **"Course" class** to store the **title**, **description** and **code** as one record.
- Using the **"Course" class**, modify the codes of the adaptor class and holder class accordingly.
- Go to "<https://www.sp.edu.sg/engineering-cluster/eee>" for all the course information and images.

-END-

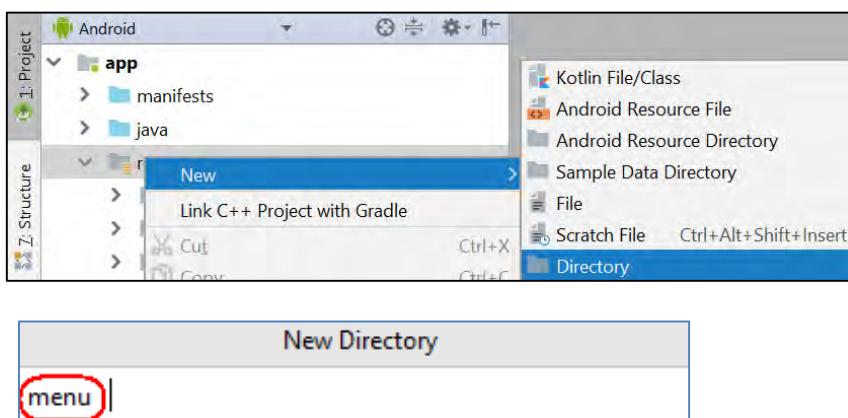
## Practical 3: Menu and SQLite Database

In this session, you will learn how to create a Menu to activate a Toast to show information entered in the restaurant detail form. In order to keep the restaurant list data persistent, you will learn how to create a database for the restaurant list using SQLite.

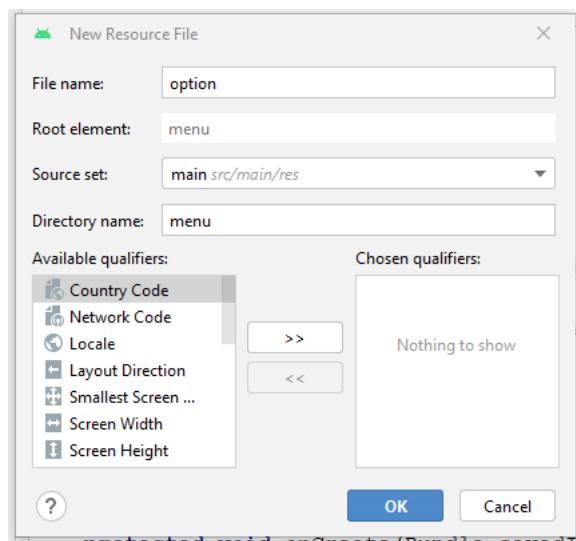
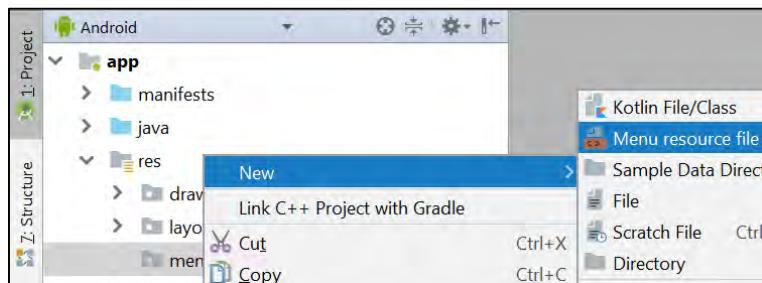


### Part I – Create Menu & Detect Menu Item Select

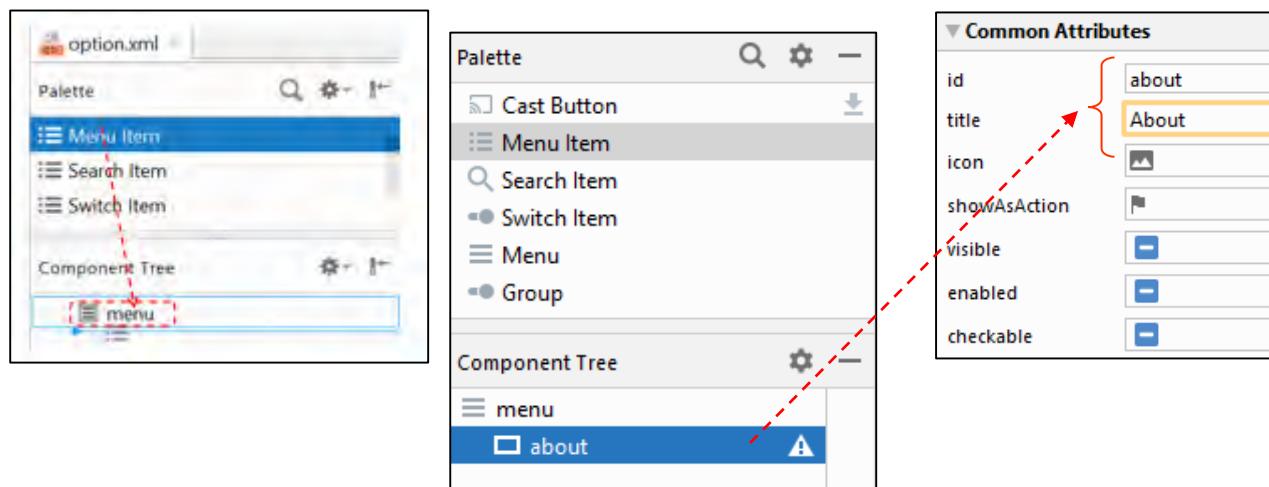
1. In this lab, we will learn how to include an image icon to a **Menu** item and detect the menu item selection.
2. Create a new project with **No Activity** and the following information:
  - **Name:** Restaurant List
  - **Package name:** com.sp.restaurantlist
  - **Save location:** C:\MAD\AndroidStudioProjects\Lab3a or D:\MAD\AndroidStudioProjects\Lab3a
  - **Language:** Java
  - **Minimum SDK:** API 27: Android 8.1 (Oreo)
  - **Source Language:** Java
  - **Build configuration language:** Groovy DSL (build.gradle)
3. Open your **Windows File Explorer** and navigate to your **Android Studio** workspace where all your projects are created
4. Copy **AndroidManifest.xml** file, **java** and **res** folders from **Lab2b\app\src\main** project folder and paste into **Lab3a\app\src\main** folder to overwrite the existing files and folders
5. At this part of the exercise, we will learn how to include an **OPTION MENU** item and limit **MENU** to be shown on '*Details*' tab only
6. At Android Studio, right click on the **res** folder and select **New > Directory** to create folder named **menu**. Hit "Enter" key.



- Right click on the newly created folder **res/menu** and select **New > Menu resource file** to create a menu file named '**option**'



- At Design Editor (**option.xml**), drag a **Menu Item** from **Palette** and drop into **menu** under Component Tree. Update the **item id** and **title** to **about** and **About** respectively.



- Open the *RestaurantList.java* file. Update the program with the following **highlighted** codes to create the **option menu** and detect **menu item selection**.

Commonly seen mistakes at this step are the spelling of these 2 methods names. Please take note.

```
public boolean onCreateOptionsMenu(Menu menu)
public boolean onOptionsItemSelected(MenuItem item)
```

Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

```
1  package com.sp.restaurantlist;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.view.LayoutInflater;
6  import android.view.Menu;
7  import android.view.MenuItem;
8  import android.view.View;
9  import android.view.ViewGroup;
10 import android.widget.AdapterView;
11 import android.widget.ArrayAdapter;
12 import android.widget.Button;
13 import android.widget.EditText;
14 import android.widget.ImageView;
15 import android.widget.ListView;
16 import android.widget.RadioGroup;
17 import android.widget.TabHost;
18 import android.widget.TextView;
19 import android.widget.Toast;
20
21 import java.util.ArrayList;
22 import java.util.List;
23
24 public class RestaurantList extends AppCompatActivity {
25     private EditText restaurantName;
26     private RadioGroup restaurantTypes;
27     private Button buttonSave;
28     private EditText restaurantAddress;
29     private EditText restaurantTel;
30
31     private List<Restaurant> model = new ArrayList<>();
32     private RestaurantAdapter adapter = null;
33     private ListView list;
34     private TabHost host;
35
36     @Override
37     protected void onCreate(Bundle savedInstanceState) {
38         super.onCreate(savedInstanceState);
39         setContentView(R.layout.main);
40
41         restaurantName = findViewById(R.id.restaurant_name);
42         restaurantTypes = findViewById(R.id.restaurant_types);
43
44         buttonSave = findViewById(R.id.button_save);
45         buttonSave.setOnClickListener(onSave);
46
47         restaurantAddress = findViewById(R.id.restaurant_address);
48         restaurantTel = findViewById(R.id.restaurant_tel);
49
50         list = findViewById(R.id.restaurants);
51         adapter = new RestaurantAdapter();
52         list.setAdapter(adapter);
53
54         host = findViewById(R.id.tabHost);
55         host.setup();
56 }
```

```
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115

//Tab 1
TabHost.TabSpec spec = host.newTabSpec("List");
spec.setContent(R.id.restaurants_tab);
spec.setIndicator("List");
host.addTab(spec);

//Tab 2
spec = host.newTabSpec("Details");
spec.setContent(R.id.details_tab);
spec.setIndicator("Details");
host.addTab(spec);
host.setCurrentTab(1);
list.setOnItemClickListener(onListClick);

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.option, menu);
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.about) {
        Toast.makeText(this, "Restaurant List - version 1.0", Toast.LENGTH_LONG).show();
    }
    return super.onOptionsItemSelected(item);
}

private View.OnClickListener onSave = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // To read data from restaurantName EditText
        String nameStr = restaurantName.getText().toString();
        // To read data from restaurantAddress EditText
        String addressStr = restaurantAddress.getText().toString();
        // To read data from restaurantTel EditText
        String telStr = restaurantTel.getText().toString();

        String restType = "";
        //To read selection of restaurantTypes RadioGroup
        int radioID = restaurantTypes.getCheckedRadioButtonId();
        if (radioID == R.id.chinese ) {
            restType = "Chinese";
        } else
        if (radioID == R.id.western ) {
            restType = "Western";
        } else
        if (radioID == R.id.indian ) {
            restType = "Indian";
        } else
        if (radioID == R.id.indonesian ) {
            restType = "Indonesian";
        } else
        if (radioID == R.id.korean) {
            restType = "Korean";
        } else
    }
}
```

Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering

```
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139 @
140
141
142
143
144 @
145
146
147
148
149
150 ⓘ
151
152
153
154
155
156
157
158
159
160
161
162 @↑
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177

    if (radioID == R.id.japanese) {
        restType = "Japanese";
    } else
        if (radioID == R.id.thai) {
            restType = "Thai";
        }
        //String combineStr = nameStr + "\n" + addressStr + "\n" + telStr + "\n" +restType;
        //Toast.makeText(v.getContext(), combineStr, Toast.LENGTH_LONG).show();
        Restaurant restaurant = new Restaurant();
        restaurant.setName(nameStr);
        restaurant.setAddress(addressStr);
        restaurant.setTelephone(telStr);
        restaurant.setRestaurantType(restType);
        adapter.add(restaurant);
        host.setCurrentTab(0);
    }

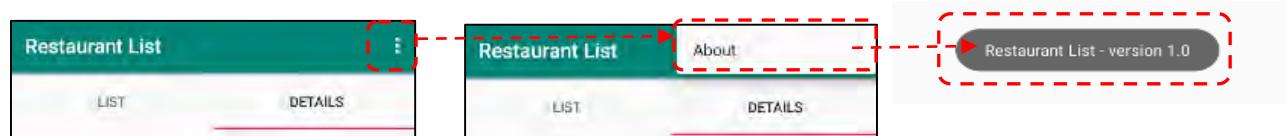
static class RestaurantHolder {
    private TextView restName = null;
    private TextView addr = null;
    private ImageView icon = null;
    RestaurantHolder(View row) {
        restName = row.findViewById(R.id.restName);
        addr = row.findViewById(R.id.restAddr);
        icon = row.findViewById(R.id.icon);
    }
    void populateFrom(Restaurant r) {
        restName.setText(r.getName());
        addr.setText(r.getAddress() + ", " + r.getTelephone());
        if (r.getRestaurantType().equals("Chinese")) {
            icon.setImageResource(R.drawable.ball_red);
        } else if (r.getRestaurantType().equals("Western")) {
            icon.setImageResource(R.drawable.ball_yellow);
        } else {
            icon.setImageResource(R.drawable.ball_green);
        }
    }
}

class RestaurantAdapter extends ArrayAdapter<Restaurant> {
    RestaurantAdapter() { super(RestaurantList.this,R.layout.row, model); }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View row = convertView;
        RestaurantHolder holder;
        if (row == null) {
            LayoutInflator inflater = getLayoutInflater();
            row = inflater.inflate(R.layout.row, parent, false);
            holder = new RestaurantHolder(row);
            row.setTag(holder);
        } else {
            holder = (RestaurantHolder)row.getTag();
        }
        holder.populateFrom(model.get(position));
        return (row);
    }
}
```

```
178
179
180
181 *↑ AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
182     @Override
183     public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
184         Restaurant r = model.get(position);
185
186         restaurantName.setText(r.getName());
187         restaurantAddress.setText(r.getAddress());
188         restaurantTel.setText(r.getTelephone());
189
190         if (r.getRestaurantType().equals("Chinese")) {
191             restaurantTypes.check(R.id.chinese);
192         } else if (r.getRestaurantType().equals("Western")) {
193             restaurantTypes.check(R.id.western);
194         } else if (r.getRestaurantType().equals("Indian")) {
195             restaurantTypes.check(R.id.indian);
196         } else if (r.getRestaurantType().equals("Indonesian")) {
197             restaurantTypes.check(R.id.indonesian);
198         } else if (r.getRestaurantType().equals("Korean")) {
199             restaurantTypes.check(R.id.korean);
200         } else if (r.getRestaurantType().equals("Japanese")) {
201             restaurantTypes.check(R.id.japanese);
202         } else {
203             restaurantTypes.check(R.id.thai);
204         }
205     }
206 }
```

10. Run the *Lab3a* project. Click on the MENU button, the **About** option menu item will pop-up. When click on the item from the option menu, a toast will be displayed.



## Further Enhancements

11. At the moment, the option menu and its “About” menu item can be activated at the “LIST” tab and “DETAILS” tab. We will now update the `onCreateOptionsMenu(Menu menu)` method to limit the “About” menu to pop-up only when user is at the “DETAILS” tab. The `onCreateOptionsMenu(Menu menu)` method is called every time when `invalidateOptionsMenu()` is called. The `onCreateOptionsMenu(Menu menu)` method must return `true` for the menu to be displayed; and return `false` to hide the menu.

**Hint:** Open the `RestaurantList` activity and update the program with the following codes:

- i. Declare an extra `boolean` variable named `showMenu`

### Declaration

```
private boolean showMenu = false;
```

- ii. Set `TabHost` a `setOnTabChangedListener` which is a listener to detect change of tab view. By calling the method `invalidateOptionsMenu()` we update `showMenu` according to current tab view selected. If the tab is at “LIST” tab, `showMenu` is set to `false`. Otherwise, `showMenu` is set to `true`.

### PART I – add within `onCreate()` method

The `invalidateOptionsMenu()` method is called when there is a change in the tab selection.

```
host.setOnTabChangedListener(new TabHost.OnTabChangeListener() {
    @Override
    public void onTabChanged(String tabId) {
        invalidateOptionsMenu();
    }
});
```

### PART II – after the `onCreate()` method add a `invalidateOptionsMenu()` callback method

```
@Override
public void invalidateOptionsMenu() {
    if (host.getCurrentTab() == 0) {
        showMenu = false;
    } else if (host.getCurrentTab() == 1) {
        showMenu = true;
    }
    super.invalidateOptionsMenu();
}
```

- iii. The `onCreateOptionsMenu(Menu menu)` method will be called automatically each time the `invalidateOptionsMenu()` method is called.

### PART III – update `onCreateOptionsMenu()` callback method

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    if (showMenu == true) {
        getMenuInflater().inflate(R.menu.option, menu);
        return true;
    }
    else
        return false;
}
```

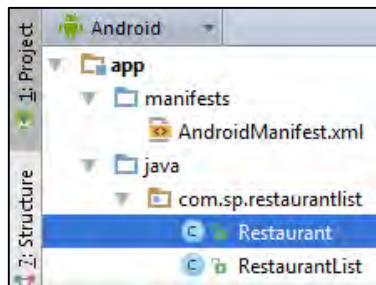
- iv. After the **onCreate()** method add a **onStart()** callback method which will call the **invalidateOptionsMenu()** method when the activity is start up.

```
@Override  
protected void onStart() {  
    invalidateOptionsMenu();  
    super.onStart();  
}
```

12. Run *Lab3a* project. Test the **MENU** display constraint. **Show to your lecturer after you have completed.**

## Part II – Using Android SQLite Database

13. At the moment, all data saved in the Restaurant List will be lost whenever the app is no more active. In this exercise, Android SQLite database will be used to hold the restaurant data. The data saved will stay persist with the app from run to run
14. Create a new project with **No Activity** and the following information:
  - **Name:** Restaurant List
  - **Package name:** com.sp.restaurantlist
  - **Save location:** C:\MAD\AndroidStudioProjects\Lab3b or D:\MAD\AndroidStudioProjects\Lab3b
  - **Language:** Java
  - **Minimum SDK:** API 27: Android 8.1 (Oreo)
  - **Source Language:** Java
  - **Build configuration language:** Groovy DSL (build.gradle)
15. Open **Windows File Explorer** and navigate to your Android Studio workspace (C:\MAD\AndroidStudioProjects or D:\MAD\AndroidStudioProjects) where all your projects are created and saved.
16. Double click to open the *Lab3a* project folder and navigate down to “app\src\main” folder. Copy **AndroidManifest.xml** file, **java** and **res** folders
17. Go to newly created project “*Lab3b*\app\src\main” folder and paste into it to overwrite existing folders and files
18. Go back to Android Studio. Open the *RestaurantList.java* and *main.xml* files and they should show the content from *Lab3b*
19. Expand the **java/com.sp.restaurantlist** folder. Right click on the *Restaurant.java* file and select **Delete** to remove the file from the project. *Restaurant.java* will be replaced with *RestaurantHelper.java* that deals with the SQLite database operations.



20. SQLite is an Open Source Database which is embedded into Android. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. More information about SQLite can be found on the SQLite website: <http://www.sqlite.org>.
21. A database is like a bookshelf, a database table is like a file folder and records (database model) saved in database table is like form kept in file folder



(i) Bookshelf → Database



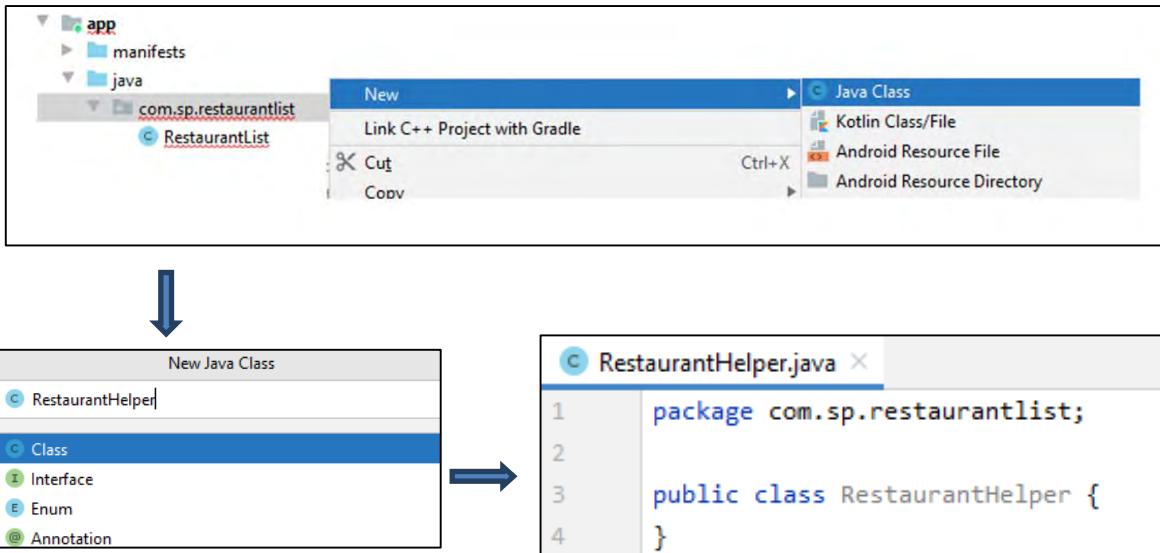
(ii) File folder → Database Table



(iii) Forms → Records in Table

22. With the basic understanding of database functionality, we will first create a class that handles all the operations required to deal with the database such as creating the database, creating tables, inserting and deleting records and so on

23. The first step is to create a class **RestaurantHelper.java** that inherits from **SQLiteOpenHelper** class.



Add “extends SQLiteOpenHelper” to the class header:

```
class RestaurantHelper extends SQLiteOpenHelper {
```

24. This class provides two methods to override to deal with the database:

- **onCreate(SQLiteDatabase db);**: invoked when the database is created, this is where we can create tables and columns to them, create views or triggers.

```
@Override
public void onCreate(SQLiteDatabase db) {
    // will be called once when database has not been created
    db.execSQL("CREATE TABLE restaurants_table ( " +
               "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
               "restaurantName TEXT, " +
               "restaurantAddress TEXT, " +
               "restaurantTel TEXT, " +
               "restaurantType TEXT);");
}
```

- **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion);**: invoked when we make a modification to the database such as altering, dropping , creating new tables.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Will not be called until SCHEMA_VERSION increases
    // Here we can upgrade the database e.g. add more tables
}
```

25. The database and table created for the Restaurant List application will have the following name and fields:

|                        |                          |
|------------------------|--------------------------|
| <b>Database's Name</b> | <b>restaurentlist.db</b> |
| <b>Table's Name</b>    | <b>restaurants_table</b> |

**restaurants\_table Format:**

| <b>Field's Name</b> | <b>Type</b>           | <b>Key</b> | <b>Description</b>                             |
|---------------------|-----------------------|------------|------------------------------------------------|
| _id                 | INTEGER AUTOINCREMENT | PRIMARY    | Create a unique integer number for each record |
| restaurantName      | TEXT                  |            |                                                |
| restaurantAddress   | TEXT                  |            |                                                |
| restaurantTel       | TEXT                  |            |                                                |
| restaurantType      | TEXT                  |            |                                                |

**Take note of the `_id`, there should be no space between the underscore (`_`) symbol and “id”.**

26. In the **RestaurantHelper** class, **onCreate()** is called by the framework, if the database does not exists. **SQLiteDatabase** provides the methods **getReadableDatabase()** and **getWritableDatabase()** to get access to a **SQLiteDatabase** object; either in read or write mode.

```

@Override
public void onCreate(SQLiteDatabase db) {
    // will be called once when database has not been created
    db.execSQL("CREATE TABLE restaurants_table ( " +
        "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "restaurantName TEXT, " +
        "restaurantAddress TEXT, " +
        "restaurantTel TEXT, " +
        "restaurantType TEXT);");
}

10 usages
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Will not be called until SCHEMA_VERSION increases
    // Here we can upgrade the database e.g. add more tables
}

```

27. Update the **RestaurantHelper.java** file with the following content and save.

- **onCreate()** – creates a database table within the database called **restaurentlist.db**  
This has to be done before any insert/select/delete SQL operations
- **getAll()** – triggers an SQL query. The return holds the result of the query
- **insert()** – inserts a new record (with restaurant info) into the database table
- **c.getString(i)** – retrieves data from index i of cursor c

All these methods are housed within the class **RestaurantHelper**.

Hence to use them, first of all a **RestaurantHelper** object must be created.

E.g.

```
RestaurantHelper helper = new RestaurantHelper(this);
helper.getAll();
```

Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering

```
1 package com.sp.restaurantlist;
2
3 import android.content.ContentValues;
4 import android.content.Context;
5 import android.database.Cursor;
6 import android.database.sqlite.SQLiteDatabase;
7 import android.database.sqlite.SQLiteOpenHelper;
8
9 public class RestaurantHelper extends SQLiteOpenHelper {
10     private static final String DATABASE_NAME = "restaurantlist.db";
11     private static final int SCHEMA_VERSION = 1;
12
13     public RestaurantHelper(Context context) {
14         super(context, DATABASE_NAME, null, SCHEMA_VERSION);
15     }
16
17     @Override
18     public void onCreate(SQLiteDatabase db) {
19         // Will be called once when the database is not created
20         db.execSQL("CREATE TABLE restaurants_table ( _id INTEGER PRIMARY KEY AUTOINCREMENT," +
21                 " restaurantName TEXT, restaurantAddress TEXT, restaurantTel TEXT, restaurantType TEXT');");
22     }
23
24     @Override
25     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
26         // Will not be called until SCHEMA_VERSION increases
27         // Here we can upgrade the database e.g. add more tables
28     }
29
30     /* Read all records from restaurants_table */
31     public Cursor getAll() {
32         return (getReadableDatabase().rawQuery(
33                 "SELECT _id, restaurantName, restaurantAddress, restaurantTel," +
34                 " restaurantType FROM restaurants_table ORDER BY restaurantName", null));
35     }
36
37     /* Write a record into restaurants_table */
38     public void insert(String restaurantName, String restaurantAddress,
39                         String restaurantTel, String restaurantType) {
40         ContentValues cv = new ContentValues();
41
42         cv.put("restaurantName", restaurantName);
43         cv.put("restaurantAddress", restaurantAddress);
44         cv.put("restaurantTel", restaurantTel);
45         cv.put("restaurantType", restaurantType);
46
47         getWritableDatabase().insert("restaurants_table", "restaurantName", cv);
48     }
49
50     @
51     public String getRestaurantName(Cursor c) {
52         return (c.getString(1));
53     }
54
55     @
56     public String getRestaurantAddress(Cursor c) {
57         return (c.getString(2));
58     }
59
60     @
61     public String getRestaurantTel(Cursor c) {
62         return (c.getString(3));
63     }
64
65     @
66     public String getRestaurantType(Cursor c) {
67         return (c.getString(4));
68     }
69 }
```

28. In the previous Labs, we use **ArrayAdapter** to bind the **ArrayList** and **ListView**. Any new restaurant data model added to ArrayAdapter, it will update both ArrayList and ListView automatically. Whereas we cannot use ArrayList and ArrayAdapter any more for using SQLite database. Instead of **ArrayList**, it is **replaced by Cursor** and **ArrayAdapter is replaced by CursorAdapter**. Therefore, **CursorAdapter** is now being used to bind the Cursor and ListView for any new restaurant record added
29. Open the *RestaurantList.java* file, remove extraneous code and simplify the code as follows. Save the file when completed.

```
1  package com.sp.restaurantlist;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.database.Cursor;
5  import android.os.Bundle;
6  import android.view.LayoutInflater;
7  import android.view.Menu;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.view.ViewGroup;
11 import android.widget.AdapterView;
12 import android.widget.Button;
13 import android.widget.EditText;
14 import android.widget.ImageView;
15 import android.widget.ListView;
16 import android.widget.RadioGroup;
17 import android.widget.TabHost;
18 import android.widget.TextView;
19 import android.widget.Toast;
20
21 import android.content.Context;
22 import androidx.cursoradapter.widget.CursorAdapter;
23
24 public class RestaurantList extends AppCompatActivity {
25     private EditText restaurantName;
26     private RadioGroup restaurantTypes;
27     private Button buttonSave;
28     private EditText restaurantAddress;
29     private EditText restaurantTel;
30
31     private Cursor model = null;
32     private RestaurantAdapter adapter = null;
33     private ListView list;
34     private RestaurantHelper helper = null;
35     private TabHost host;
36     private boolean showMenu = false;
37
38     @Override
39     protected void onCreate(Bundle savedInstanceState) {
40         super.onCreate(savedInstanceState);
41         setContentView(R.layout.main);
42
43         restaurantName = findViewById(R.id.restaurant_name);
44         restaurantTypes = findViewById(R.id.restaurant_types);
45
46         buttonSave = findViewById(R.id.button_save);
47         buttonSave.setOnClickListener(onSave);
48
49         restaurantAddress = findViewById(R.id.restaurant_address);
50         restaurantTel = findViewById(R.id.restaurant_tel);
```

```
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77 ④↑
78
79
80
81
```

```
helper = new RestaurantHelper(this);
list = findViewById(R.id.restaurants);
model = helper.getAll();
adapter = new RestaurantAdapter(this, model, 0);
list.setAdapter(adapter);

host = findViewById(R.id.tabHost);
host.setup();

//Tab 1
TabHost.TabSpec spec = host.newTabSpec("List");
spec.setContent(R.id.restaurants_tab);
spec.setIndicator("List");
host.addTab(spec);

//Tab 2
spec = host.newTabSpec("Details");
spec.setContent(R.id.details_tab);
spec.setIndicator("Details");
host.addTab(spec);
host.setCurrentTab(1);
list.setOnItemClickListener(onListClick);

host.setOnTabChangedListener(new TabHost.OnTabChangeListener() {
    @Override
    public void onTabChanged(String tabId) {
        invalidateOptionsMenu();
    }
});
```

```
82
83
84  ↗ @Override
85  ↗     protected void onDestroy() {
86  ↗         helper.close();
87  ↗         super.onDestroy();
88  ↗     }
89
90  ↗ @Override
91  ↗     protected void onStart() {
92  ↗         invalidateOptionsMenu();
93  ↗         super.onStart();
94  ↗     }
95
96  ↗ @Override
97  ↗     public void invalidateOptionsMenu() {
98  ↗         if (host.getCurrentTab() == 0) {
99  ↗             showMenu = false;
100  ↗         } else if (host.getCurrentTab() == 1) {
101  ↗             showMenu = true;
102  ↗         }
103  ↗         super.invalidateOptionsMenu();
104  ↗     }
105  ↗ @Override
106  ↗     public boolean onCreateOptionsMenu(Menu menu) {
107  ↗         if (showMenu == true) {
108  ↗             getMenuInflater().inflate(R.menu.option, menu);
109  ↗             return true;
110  ↗         }
111  ↗         else
112  ↗             return false;
113  ↗     }
114
115  ↗ @Override
116  ↗     public boolean onOptionsItemSelected(MenuItem item) {
117  ↗         if (item.getItemId() == R.id.about)
118  ↗         {
119  ↗             Toast.makeText(this, "Restaurant List - version 1.0", Toast.LENGTH_LONG).show();
120  ↗             break;
121  ↗         }
122  ↗         return super.onOptionsItemSelected(item);
123  ↗     }
124
125
126  ↗ private View.OnClickListener onSave = new View.OnClickListener() {
127  ↗     @Override
128  ↗     public void onClick(View v) {
129  ↗         // To read data from restaurantName EditText
130  ↗         String nameStr = restaurantName.getText().toString();
131  ↗         // To read data from restaurantAddress EditText
132  ↗         String addressStr = restaurantAddress.getText().toString();
133  ↗         // To read data from restaurantTel EditText
134  ↗         String telStr = restaurantTel.getText().toString();
135  ↗         String restType = "";
136  ↗         //To read selection of restaurantTypes RadioGroup
137  ↗         int radioID = restaurantTypes.getCheckedRadioButtonId();
138  ↗         if (radioID == R.id.chinese ) {
139  ↗             restType = "Chinese";
140  ↗         } else
141  ↗         if (radioID == R.id.western ) {
```

```
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192

    if (radioID == R.id.indian) {
        restType = "Indian";
    } else
    if (radioID == R.id.indonesian) {
        restType = "Indonesian";
    } else
    if (radioID == R.id.korean) {
        restType = "Korean";
    } else
    if (radioID == R.id.japanese) {
        restType = "Japanese";
    } else
    if (radioID == R.id.thai) {
        restType = "Thai";
    }
    //Insert record into SQLite table
    helper.insert(nameStr, addressStr, telStr, restType);

    model = helper.getAll();      //Update Cursor after new record is added
    adapter.swapCursor(model);
    host.setCurrentTab(0);
}

};

static class RestaurantHolder {
    private TextView restName = null;
    private TextView addr = null;
    private ImageView icon = null;

    RestaurantHolder(View row) {
        restName = row.findViewById(R.id.restName);
        addr = row.findViewById(R.id.restAddr);
        icon = row.findViewById(R.id.icon);
    }

    void populateFrom(Cursor c, RestaurantHelper helper) {
        restName.setText(helper.getRestaurantName(c));
        String temp = helper.getRestaurantAddress(c) + ", " + helper.getRestaurantTel(c);
        addr.setText(temp);

        if (helper.getRestaurantType(c).equals("Chinese")) {
            icon.setImageResource(R.drawable.ball_red);
        } else if (helper.getRestaurantType(c).equals("Western")) {
            icon.setImageResource(R.drawable.ball_yellow);
        } else {
            icon.setImageResource(R.drawable.ball_green);
        }
    }
}
```

Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering

```
193
194
195
196
197
198
199 *| @
200
201
202
203
204
205 *| |
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240

class RestaurantAdapter extends CursorAdapter {
    RestaurantAdapter(Context context, Cursor cursor, int flags) {
        super(context, cursor, flags);
    }

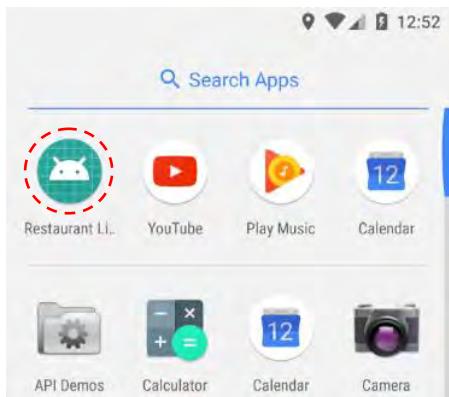
    @Override
    public void bindView(View view, Context context, Cursor cursor) {
        RestaurantHolder holder = (RestaurantHolder) view.getTag();
        holder.populateFrom(cursor, helper);
    }

    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent) {
        LayoutInflator inflater = getLayoutInflator();
        View row = inflater.inflate(R.layout.row, parent, false);
        RestaurantHolder holder = new RestaurantHolder(row);
        row.setTag(holder);
        return (row);
    }
}

AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        model.moveToPosition(position);
        restaurantName.setText(helper.getRestaurantName(model));
        restaurantAddress.setText(helper.getRestaurantAddress(model));
        restaurantTel.setText(helper.getRestaurantTel(model));

        if (helper.getRestaurantType(model).equals("Chinese")) {
            restaurantTypes.check(R.id.chinese);
        } else if (helper.getRestaurantType(model).equals("Western")) {
            restaurantTypes.check(R.id.western);
        } else if (helper.getRestaurantType(model).equals("Indian")) {
            restaurantTypes.check(R.id.indian);
        } else if (helper.getRestaurantType(model).equals("Indonesian")) {
            restaurantTypes.check(R.id.indonesian);
        } else if (helper.getRestaurantType(model).equals("Korean")) {
            restaurantTypes.check(R.id.korean);
        } else if (helper.getRestaurantType(model).equals("Japanese")) {
            restaurantTypes.check(R.id.japanese);
        } else {
            restaurantTypes.check(R.id.thai);
        }
        host.setCurrentTab(1);
    };
};
```

**Note: If you have any errors with the SQLite database, you need to uninstall the Restaurant List App before testing again.**



30. Run the *Lab3b* project. Enter a restaurant data and save. Click on the back button to exit from the app
31. Click on the **Restaurant List** App to run the app again. The previously saved data will stay on the list
32. **If you have completed the project, demo it to your lecturer**

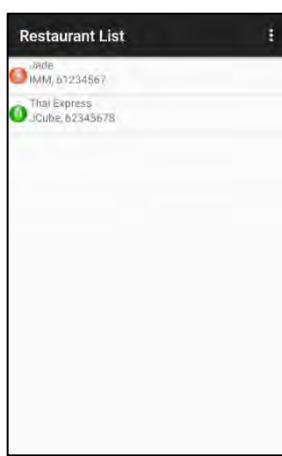
**-END-**

## Practical 4: Activities

In this session, you will learn to make use of Explicit Intent to activate ‘Details’ Form activity, edit & update the record saved in SQLite table “restaurants\_table”.

### Part I – From Single Activity to Two Activities

1. In previous lab exercises, we have successfully split the **UI Views** into two ('List' & 'Details') using tabs. The controllers for these UI views is still **using a single activity class (class RestaurantList)**.
2. For better way of managing an Android app, it will be **one controller for one UI View within one activity**. Hence, in this part of the exercise, we will reorganize the **UI View with individual activity controller and layout file**:
  - ‘List’ – **class RestaurantList (ListActivity)** and **main.xml** layout
  - ‘Details’ form – **class DetailForm (AppCompatActivity)** and **detail\_form.xml** layout



List activity (launcher)  
File: **RestaurantList.java**  
Layout: **main.xml**  
Display: list of restaurants previously in List tab



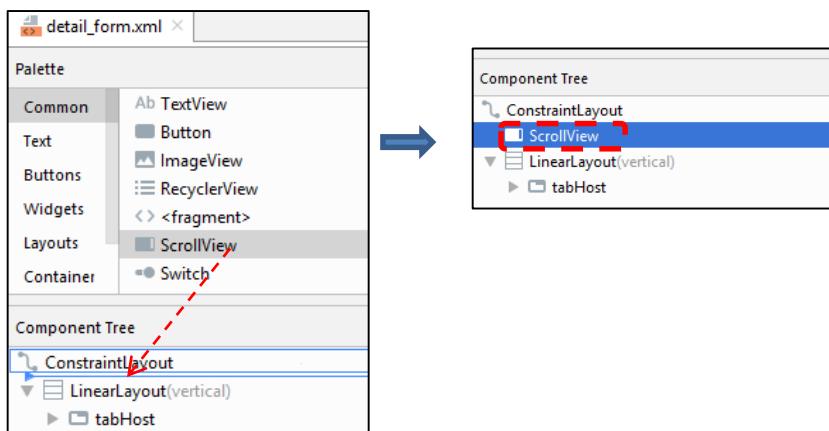
Details form activity (2<sup>nd</sup> activity)  
File: **DetailForm.java**  
Layout: **detail\_form.xml**  
Display: details form previously in Details tab

3. Create a new project with **No Activity** and the following information:
  - **Name:** *Restaurant List*
  - **Package name:** *com.sp.restaurantlist*
  - **Save location:** *C:\MAD\AndroidStudioProjects\Lab4 or D:\MAD\AndroidStudioProjects\Lab4*
  - **Language:** *Java*
  - **Minimum SDK:** *API 27: Android 8.1 (Oreo)*
  - **Source Language:** **Java**
  - **Build configuration language:** **Groovy DSL (build. grade)**
4. Open your **Windows File Explorer** and navigate to your **Android Studio** workspace where all your projects are created.
5. Copy *AndroidManifest.xml* file, **java** and **res** folders from **Lab3b\app\src\main** project folder and paste into **Lab4\app\src\main** folder to overwrite the existing file and folder
6. At Windows Explorer, navigate to **Lab4\app\src\main\res\layout** folder. Copy the *main.xml* file and paste into the same folder to create a duplicated copy
7. Right click on **main – Copy.xml** file and rename it to **detail\_form.xml**

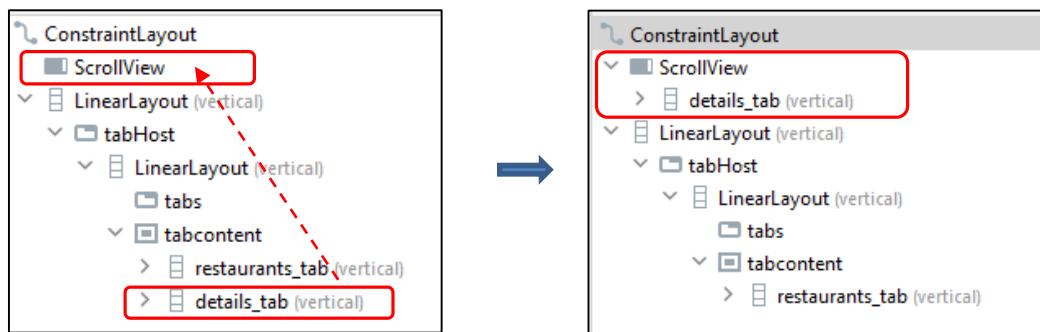
### Splitting UI views & Change MENU Item

8. For *detail\_form.xml*, we are going to keep only the UI previously in Details tab.

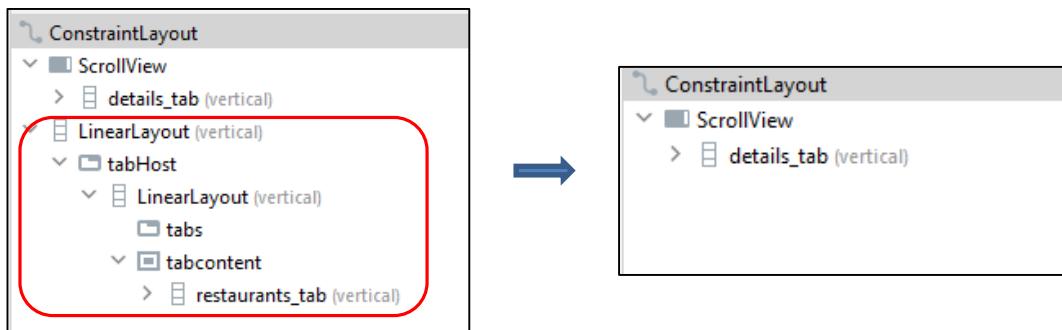
At Android Studio, open the *detail\_form.xml* from the **res/layout** folder. At the **Design Editor > Palette** pane, drag a **ScrollView** widget and drop into the **ConstraintLayout** widget as a child node.



9. Drag the **details\_tab** widget and drop into **ScrollView** widget.

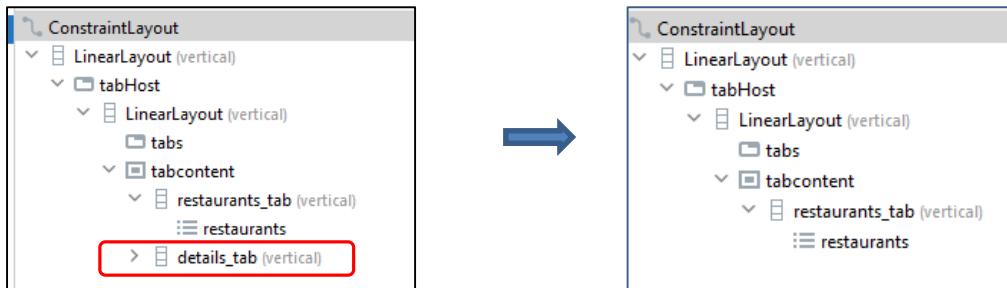


10. Delete the **LinearLayout** from the **ConstraintLayout** child node

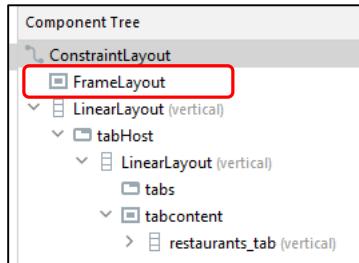


11. For the *main.xml*, we are going to keep only the UI previously in List tab.

Open the *main.xml* layout file. Delete **details\_tab**



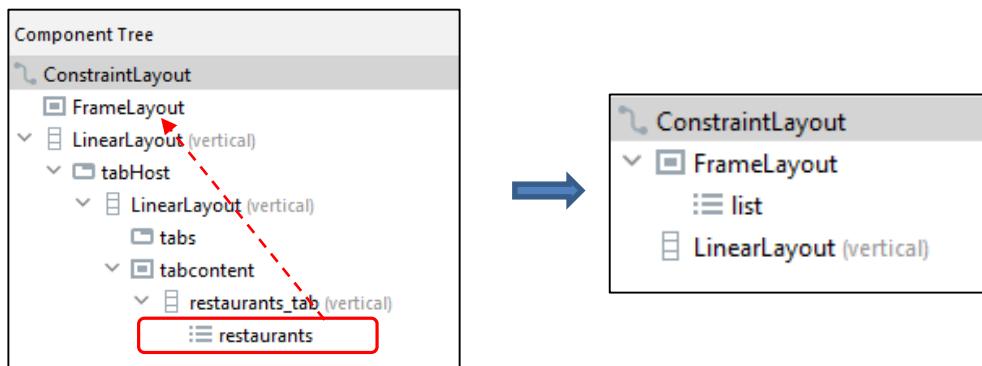
12. Drag and drop the **FrameLayout** widget into **ConstraintLayout** of *main.xml* file.



13. Drag the **restaurants** (**ListView**) widget and drop into the **FrameLayout**.

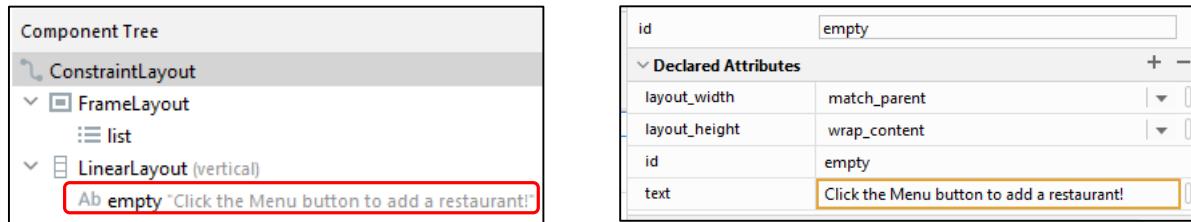
Change id from **restaurants** → **list**.

Delete the whole **tabHost** widget.

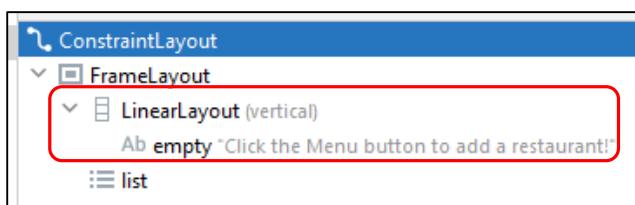


14. Drag a **TextView** widget from **Palette** pane and into the **LinearLayout** widget.

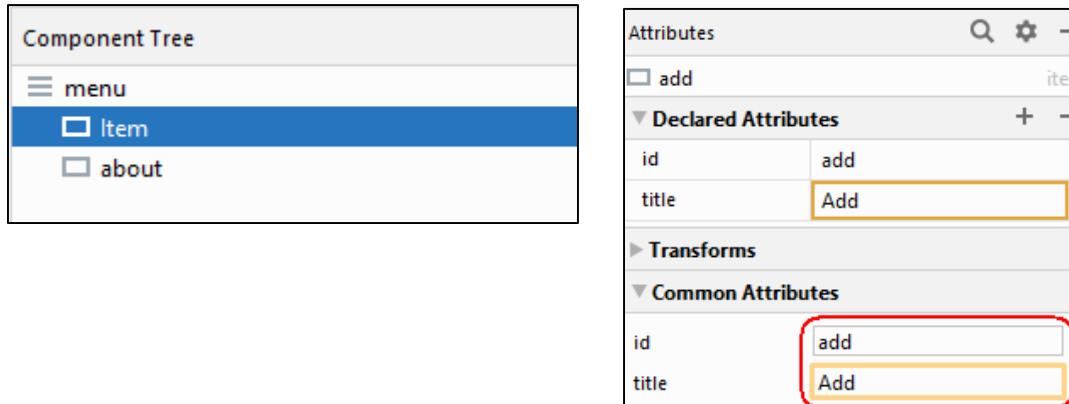
Update the **TextView ID** to 'empty' and text as '**Click the MENU button to add a restaurant!**'.



15. Drag the **LinearLayout (vertical)** child node and drop into the **FrameLayout**

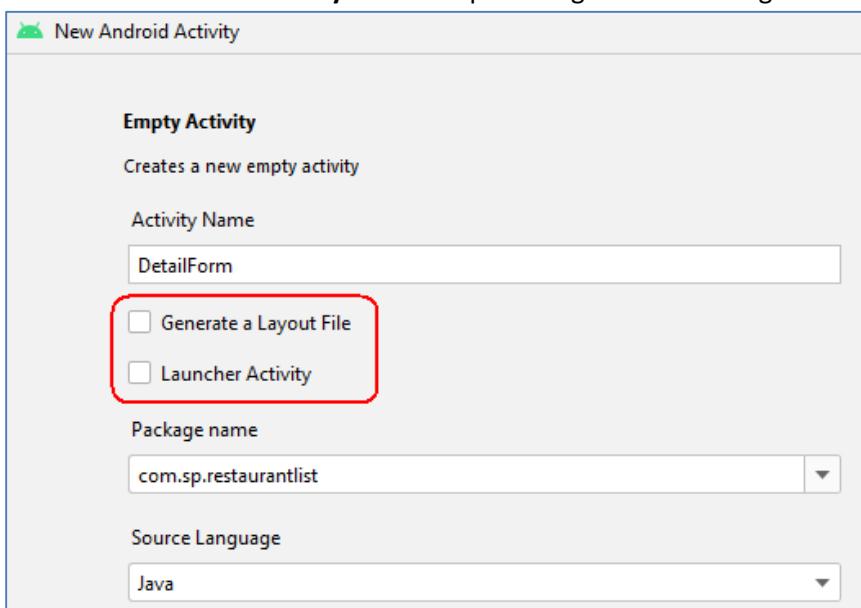


16. Open the *option.xml* file in the **res > menu** folder, drag in a new menu item and change the menu item **id** and **title** to “**add**” and “**Add**” respectively.



### Creating Separate Activities

17. At the moment, the program has only 1 activity, ***RestaurantList.java***. It is going to be responsible for displaying only the ‘List’ UI. We are going to create a 2<sup>nd</sup> activity to handle the ‘Details’ form UI.  
 18. Right click on the **java/com.sp.restaurantlist** folder, select **New > Activity > Empty Views Activity** to create **DetailForm.java** controller to manage the ‘Details’ form UI view.  
 Uncheck the “Generate a Layout File” option to get the following:



19. Make sure that “Generate Layout File” and “Launcher Activity” are unchecked.
20. The function of the *DetailForm.java* Activity is:
1. to load the *detail\_form.xml* layout as the user’s UI view using `setContentView(R.layout.detail_form)` method.
  2. to link up to the widgets on UI view using `findViewById` method.
  3. to capture the data entries of each widgets using `getText().toString()` method
  4. to create a new record in the SQLite *restaurants\_table* using the `insert` method provided by the *RestaurantHelper* (*SQLiteOpenHelper* subclass).

21. Update the *DetailForm.java* file with the following content and save

(Note: Most of the code can be transferred from *RestaurantList.java* file)

```
1 package com.sp.restaurantlist;
2
3 import android.os.Bundle;
4 import android.view.View;
5 import android.widget.Button;
6 import android.widget.EditText;
7 import android.widget.RadioGroup;
8
9 import androidx.appcompat.app.AppCompatActivity;
10
11 public class DetailForm extends AppCompatActivity {
12     private EditText restaurantName;
13     private EditText restaurantAddress;
14     private EditText restaurantTel;
15     private RadioGroup restaurantTypes;
16     private Button buttonSave;
17
18     private RestaurantHelper helper = null;
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.detail_form);
24         restaurantName = findViewById(R.id.restaurant_name);
25         restaurantAddress = findViewById(R.id.restaurant_address);
26         restaurantTel = findViewById(R.id.restaurant_tel);
27         restaurantTypes = findViewById(R.id.restaurant_types);
28
29         buttonSave = findViewById(R.id.button_save);
30         buttonSave.setOnClickListener(onSave);
31         helper = new RestaurantHelper(this);
32     }
33
34     @Override
35     protected void onDestroy() {
36         super.onDestroy();
37         helper.close();
38     }
39
40     View.OnClickListener onSave = new View.OnClickListener() {
41         @Override
42         public void onClick(View v) {
43             // To read date from EditText
44             String nameStr = restaurantName.getText().toString();
45             String addrStr = restaurantAddress.getText().toString();
46             String telStr = restaurantTel.getText().toString();
47             String restType = "";
48             //To read selection of restaurantTypes RadioGroup
49             int radioID = restaurantTypes.getCheckedRadioButtonId();
50             if (radioID == R.id.chinese) {
51                 restType = "Chinese";
52             } else
53                 if (radioID == R.id.western) {
54                     restType = "Western";
55                 } else
56                     if (radioID == R.id.indian) {
57                         restType = "Indian";
58                     } else
```

```
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```

```
if (radioID == R.id.indonesian) {
    restType = "Indonesian";
} else
if (radioID == R.id.korean) {
    restType = "Korean";
} else
if (radioID == R.id.japanese) {
    restType = "Japanese";
} else
if (radioID == R.id.thai) {
    restType = "Thai";
}

helper.insert(nameStr, addrStr, telStr, restType);

//To close current Activity class and exit
finish();
};

}
```

22. With the 'Details' form UI view controller done, we will look into the 'List' UI view controller (*RestaurantList.java*)
23. The function of *RestaurantList.java* is:
1. to load the *main.xml* layout as user's UI view.
  2. to display record(s) saved in *restaurants\_table* in the *ListView* using *CursorAdapter*.
  3. to provide an 'Add' MENU item to make an **Explicit Intent** call to launch *DetailForm.java* activity
24. Update the *RestaurantList.java* with the following content and save

```
1   package com.sp.restaurantlist;
2
3   import android.content.Context;
4   import android.content.Intent;
5   import android.database.Cursor;
6   import android.os.Bundle;
7   import android.view.LayoutInflater;
8   import android.view.Menu;
9   import android.view.MenuItem;
10  import android.view.View;
11  import android.view.ViewGroup;
12  import android.widget.ImageView;
13  import android.widget.ListView;
14  import android.widget.TextView;
15
16  import androidx.appcompat.app.AppCompatActivity;
17  import androidx.cursoradapter.widget.CursorAdapter;
18
19  public class RestaurantList extends AppCompatActivity {
20      private Cursor model = null;
21      private RestaurantAdapter adapter = null;
22      private ListView list;
23      private RestaurantHelper helper = null;
24      private TextView empty = null;
25
```

Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering

```
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.main);
31
32         empty = findViewById(R.id.empty);
33         helper = new RestaurantHelper(this);
34         list = findViewById(R.id.list);
35         model = helper.getAll();
36         adapter = new RestaurantAdapter(this, model, 0);
37         list.setAdapter(adapter);
38     }
39
40     @Override
41     protected void onResume() {
42         super.onResume();
43         if (model != null) {
44             model.close();
45         }
46         model = helper.getAll();
47         if (model.getCount() > 0) {
48             empty.setVisibility(View.INVISIBLE);
49         }
50         adapter.swapCursor(model);
51     }
52
53     @Override
54     protected void onDestroy() {
55         helper.close();
56         super.onDestroy();
57     }
```

```
57
58     @Override
59     public boolean onCreateOptionsMenu(Menu menu) {
60         getMenuInflater().inflate(R.menu.option, menu);
61         return super.onCreateOptionsMenu(menu);
62     }
63
64     @Override
65     public boolean onOptionsItemSelected(MenuItem item) {
66         Intent intent;
67         if (item.getItemId() == R.id.add) {
68             intent = new Intent(RestaurantList.this, DetailForm.class);
69             startActivity(intent);
70         }
71         return super.onOptionsItemSelected(item);
72     }
73
74
75
76     static class RestaurantHolder {
77         private TextView restName = null;
78         private TextView addr = null;
79         private ImageView icon = null;
80
81         @ RestaurantHolder(View row) {
82             restName = row.findViewById(R.id.restName);
83             addr = row.findViewById(R.id.restAddr);
84             icon = row.findViewById(R.id.icon);
85         }
86
87         void populateFrom(Cursor c, RestaurantHelper helper) {
88             restName.setText(helper.getRestaurantName(c));
89             String temp = helper.getRestaurantAddress(c) + ", " + helper.getRestaurantTel(c);
90             addr.setText(temp);
91
92             if (helper.getRestaurantType(c).equals("Chinese")) {
93                 icon.setImageResource(R.drawable.ball_red);
94             } else if (helper.getRestaurantType(c).equals("Western")) {
95                 icon.setImageResource(R.drawable.ball_yellow);
96             } else {
97                 icon.setImageResource(R.drawable.ball_green);
98             }
99         }
100    }
101
102    class RestaurantAdapter extends CursorAdapter {
103        RestaurantAdapter(Context context, Cursor cursor, int flags) {
104            super(context, cursor, flags);
105        }
106
107        @Override
108        public void bindView(View view, Context context, Cursor cursor) {
109            RestaurantHolder holder = (RestaurantHolder) view.getTag();
110            holder.populateFrom(cursor, helper);
111        }
112    }
```

```

113
114     }
115
116
117
118
119
120     }
121
122 }

```

25. When **MENU** button is pressed at the ‘List’ UI view, a menu with ‘**Add**’ item will pop-up.
26. If the ‘**Add**’ menu item is selected, the item selected event generated will be captured by the `onOptionsItemSelected(MenuItem item)` method. The `item.getItemId()` method will read in the item which the user has clicked on the phone screen. In this case, it is “Add”. After comparing the **id** returned, if it is `R.id.add`, an **Explicit Intent** call will be used to start (`startActivity`) a new user interface (UI) - the *DetailForm.java* Activity

```

@Override
public boolean onOptionsItemSelected(MenuItem item){
    Intent intent;
    if (item.getItemId() == R.id.add) {
        intent = new Intent(RestaurantList.this, DetailForm.class);
        startActivity(intent);
    }
    return super.onOptionsItemSelected(item);
}

```

27. **\*IMPORTANT – new Activity (DetailForm.java) has been added**

In order for the system to recognize a newly added **Activity**, the *DetailForm.java* **Activity** need to be recorded in the *AndroidManifest.xml* file, otherwise, an error will occur whenever the ‘Add’ menu item is selected during runtime. Check that **line 15 - 17** are added to the *AndroidManifest.xml* file. If not add the line and save.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools" >
4
5      <application
6          android:allowBackup="true"
7          android:dataExtractionRules="@xml/data_extraction_rules"
8          android:fullBackupContent="@xml/backup_rules"
9          android:icon="@mipmap/ic_launcher"
10         android:label="Restaurant List"
11         android:roundIcon="@mipmap/ic_launcher_round"
12         android:supportsRtl="true"
13         android:theme="@style/Theme.RestaurantList"
14         tools:targetApi="31" >
15             <activity
16                 android:name=".DetailForm"
17                 android:exported="false" />
18             <activity
19                 android:name=".RestaurantList"
20                 android:exported="true" >
21                 <intent-filter>
22                     <action android:name="android.intent.action.MAIN" />
23                     <category android:name="android.intent.category.LAUNCHER" />
24                 </intent-filter>
25             </activity>
26         </application>

```

28. Run the *Lab4* project. Test the persistence data storage by adding a new restaurant to the list and save.
- 29. Show the result to your lecturer when you have completed this section successfully.**
30. Notice that the UI view will be switched to ‘List’ view automatically after the ‘Save’ button is pressed. Which command in the program causes such effect? Record down the Line number and the command from your program

**Java Program Name:** \_\_\_\_\_

**Line No.:** \_\_\_\_\_

**Command:** \_\_\_\_\_

## Part II – Passing Values between Activities using Intent putExtra

31. We have successfully split the UI views and controllers with separate layout files and activities. User can save restaurant record to *restaurants\_table* and update the ‘List’.
32. What about reading a record from the *restaurants\_table* when a ‘List’ item is selected and display the result in the ‘Details’ form?
33. Looking into the process flow, the UI view will switch from ‘List’ to ‘Details’ form when a list item is selected i.e. there is a change in activity too. For the ‘Details’ form to know which list item has been selected, the ‘List’ controller passes the “ID” as string data to ‘Details’ form controller using *putExtra* method. The *DetailForm* controller will use *getIntent().getStringExtra("ID")* method to read the value passed over

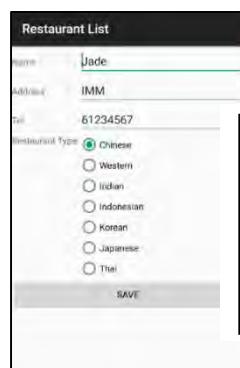


**Activity – RestaurantList.java**

```
private AdapterView.OnItemClickListener onListClick = new
    AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            model.moveToPosition(position);
            String recordID = helper.getID(model);
            Intent intent;
            intent = new Intent(RestaurantList.this, DetailForm.class);

            intent.putExtra("ID", recordID);

            startActivity(intent);
        }
    };
}
```



**Activity – DetailForm.java**

```
restaurantID = getIntent().getStringExtra("ID");

if (restaurantID != null){
    load();
}
```

34. Edit the *RestaurantHelper.java* to allow user to get record by “ID” and update the existing restaurant data.

```
1 package com.sp.restaurantlist;
2
3 import android.content.ContentValues;
4 import android.content.Context;
5 import android.database.Cursor;
6 import android.database.sqlite.SQLiteDatabase;
7 import android.database.sqlite.SQLiteOpenHelper;
8
9 public class RestaurantHelper extends SQLiteOpenHelper {
10     private static final String DATABASE_NAME = "restaurantlist.db";
11     private static final int SCHEMA_VERSION = 1;
12
13     public RestaurantHelper(Context context) {
14         super(context, DATABASE_NAME, null, SCHEMA_VERSION);
15     }
}
```

```
16
17
18 * @Override
19     public void onCreate(SQLiteDatabase db) {
20         // Will be called once when the database is not created
21         db.execSQL("CREATE TABLE restaurants_table ( _id INTEGER PRIMARY KEY AUTOINCREMENT, " +
22             "restaurantName TEXT, restaurantAddress TEXT, " +
23             "restaurantTel TEXT, restaurantType TEXT);");
24
25     * @Override
26     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
27         // Will not be called until SCHEMA_VERSION increases
28         // Here we can upgrade the database e.g. add more tables
29     }
30
31     /* Read all records from restaurants_table */
32     public Cursor getAll() {
33         return (getReadableDatabase().rawQuery(
34             "SELECT _id, restaurantName, restaurantAddress, restaurantTel, " +
35             "restaurantType FROM restaurants_table ORDER BY restaurantName", null));
36     }
37
38     /* Read a particular record from restaurants_table with id provided */
39     public Cursor getById(String id) {
40         String[] args = {id};
41
42         return (getReadableDatabase().rawQuery(
43             "SELECT _id, restaurantName, restaurantAddress, restaurantTel, " +
44             "restaurantType FROM restaurants_table WHERE _ID = ?", args));
45     }
46
47     /* Write a record into restaurants_table */
48     public void insert(String restaurantName, String restaurantAddress,
49                         String restaurantTel, String restaurantType) {
50         ContentValues cv = new ContentValues();
51
52         cv.put("restaurantName", restaurantName);
53         cv.put("restaurantAddress", restaurantAddress);
54         cv.put("restaurantTel", restaurantTel);
55         cv.put("restaurantType", restaurantType);
56
57         getWritableDatabase().insert("restaurants_table", "restaurantName", cv);
58     }
59
60     /* Update a particular record in restaurants_table with id provided */
61     public void update(String id, String restaurantName, String restaurantAddress,
62                         String restaurantTel, String restaurantType) {
63         ContentValues cv = new ContentValues();
64         String[] args = {id};
65         cv.put("restaurantName", restaurantName);
66         cv.put("restaurantAddress", restaurantAddress);
67         cv.put("restaurantTel", restaurantTel);
68         cv.put("restaurantType", restaurantType);
69
70         getWritableDatabase().update("restaurants_table", cv, " _ID = ?", args);
71     }
72 }
```

```

73     /* Read a record id value from restaurants_table */
74     @Override
75     public String getID(Cursor c) { return (c.getString(0)); }
76
77     @Override
78     public String getRestaurantName(Cursor c) { return (c.getString(1)); }
79
80     @Override
81     public String getRestaurantAddress(Cursor c) { return (c.getString(2)); }
82
83     @Override
84     public String getRestaurantTel(Cursor c) { return (c.getString(3)); }
85
86     @Override
87     public String getRestaurantType(Cursor c) { return (c.getString(4)); }
88
89   }

```

35. Open the *RestaurantList.java* file to add in the `onListItemClick(ListView list, View view, int position, long id)` method to capture the list item click event and start the **Explicit Intent** call to switch to *DetailForm.java* Activity with the 'ID' passing over.

- a) Add the highlighted code to the ***RestaurantList*** class.

```

66     @Override
67     public boolean onOptionsItemSelected(MenuItem item) {
68       if (item.getItemId() == R.id.add)
69       {
70         Intent intent;
71         intent = new Intent(RestaurantList.this, DetailForm.class);
72         startActivity(intent);
73         break;
74       }
75       return super.onOptionsItemSelected(item);
76     }
77
78     private AdapterView.OnItemClickListener onListClick = new AdapterView.OnItemClickListener() {
79       public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
80         model.moveToPosition(position);
81         String recordID = helper.getID(model);
82         Intent intent;
83         intent = new Intent(RestaurantList.this, DetailForm.class);
84         intent.putExtra("ID", recordID);
85         startActivity(intent);
86       }
87     };
88
89     static class RestaurantHolder {
90       private TextView restName = null;

```

- b) Import the ***AdapterView*** class.

```

11     import android.view.ViewGroup;
12     import android.widget.AdapterView;
13     import android.widget.ImageView;

```

- c) Add the highlighted statement to the ***onCreate*** method.

```

27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29       super.onCreate(savedInstanceState);
30       setContentView(R.layout.main);
31
32       empty = findViewById(R.id.empty);
33       helper = new RestaurantHelper(this);
34       list = findViewById(R.id.list);
35       model = helper.getAll();
36       adapter = new RestaurantAdapter(this, model, 0);
37       list.setOnItemClickListener(onListClick);
38       list.setAdapter(adapter);
39     }

```

36. Update the **DetailForm.java** file with the following content and save.

- a) Add a String variable "**restaurantID**" to **DetailForm** class.

```
12 public class DetailForm extends AppCompatActivity {
13     private EditText restaurantName;
14     private EditText restaurantAddress;
15     private EditText restaurantTel;
16     private RadioGroup restaurantTypes;
17     private Button buttonSave;
18
19     private RestaurantHelper helper = null;
20     private String restaurantID = "";
21 }
```

- b) Add the highlighted method to **DetailForm** class.

```
42 protected void onDestroy() {
43     super.onDestroy();
44     helper.close();
45 }
46
47 private void load() {
48     Cursor c = helper.getById(restaurantID);
49     c.moveToFirst();
50     restaurantName.setText(helper.getRestaurantName(c));
51     restaurantAddress.setText(helper.getRestaurantAddress(c));
52     restaurantTel.setText(helper.getRestaurantTel(c));
53
54     if (helper.getRestaurantType(c).equals("Chinese")) {
55         restaurantTypes.check(R.id.chinese);
56     } else if (helper.getRestaurantType(c).equals("Western")) {
57         restaurantTypes.check(R.id.western);
58     } else if (helper.getRestaurantType(c).equals("Indian")) {
59         restaurantTypes.check(R.id.indian);
60     } else if (helper.getRestaurantType(c).equals("Indonesian")) {
61         restaurantTypes.check(R.id.indonesian);
62     } else if (helper.getRestaurantType(c).equals("Korean")) {
63         restaurantTypes.check(R.id.korean);
64     } else if (helper.getRestaurantType(c).equals("Japanese")) {
65         restaurantTypes.check(R.id.japanese);
66     } else {
67         restaurantTypes.check(R.id.thai);
68     }
69 }
70
71 View.OnClickListener onSave = new View.OnClickListener() {
72     @Override
```

- c) Import the *Cursor* class.

```
3 import android.database.Cursor;
4 import android.os.Bundle;
5 import android.view.View;
```

- d) Add the highlighted statements to the onCreate method.

```

22    @Override
23    protected void onCreate(Bundle savedInstanceState) {
24        super.onCreate(savedInstanceState);
25        setContentView(R.layout.detail_form);
26        restaurantName = findViewById(R.id.restaurant_name);
27        restaurantAddress = findViewById(R.id.restaurant_address);
28        restaurantTel = findViewById(R.id.restaurant_tel);
29        restaurantTypes = findViewById(R.id.restaurant_types);
30
31        buttonSave = findViewById(R.id.button_save);
32        buttonSave.setOnClickListener(onSave);
33        helper = new RestaurantHelper(this);
34
35        restaurantID = getIntent().getStringExtra("ID");
36        if (restaurantID != null) {
37            load();
38        }
39    }

```

- e) Modify the code for “**View.OnClickListener onSave**” object.

```

View.OnClickListener onSave = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // To read date from EditText
        String nameStr = restaurantName.getText().toString();
        String addrStr = restaurantAddress.getText().toString();
        String telStr = restaurantTel.getText().toString();
        String restType = "";
        //To read selection of restaurantTypes RadioGroup
        int radioID = restaurantTypes.getCheckedRadioButtonId();
        if (radioID == R.id.chinese ) {
            restType = "Chinese";
        } else
        if (radioID == R.id.western ) {
            restType = "Western";
        } else
        if (radioID == R.id.indian ) {
            restType = "Indian";
        } else
        if (radioID == R.id.indonesian ) {
            restType = "Indonesian";
        } else
        if (radioID == R.id.korean) {
            restType = "Korean";
        } else
        if (radioID == R.id.japanese) {
            restType = "Japanese";
        } else
        if (radioID == R.id.thai) {
            restType = "Thai";
        }

        if (restaurantID == null) {
            helper.insert(nameStr, addrStr, telStr, restType);
        } else {
            helper.update(restaurantID, nameStr, addrStr, telStr, restType);
        }

        //To close current Activity class and exit
        finish();
    }
};

```

37. When the controller is switched to *DetailForm.java* Activity, the value of the record ‘ID’ that is passed over and will be read using the **getIntent** method and save to the local variable **restaurantID**

```
restaurantID = getIntent().getStringExtra("ID");
```

38. How does the *DetailForm.java* Activity differentiate if it is supposed

- to **read a record** from the *restaurants\_table* and **display** in the ‘Details’ form, or
- to **provide an empty ‘Details’ form** for user to **add a new restaurant record?**

If the intention is to add a new record, the value of “*restaurantID*” will be null. Otherwise, the record will be retrieved through the **load()** method call

```
private void load() {  
    Cursor c = helper.get GetById(restaurantID);  
    c.moveToFirst();  
    restaurantName.setText(helper.getRestaurantName(c));  
    restaurantAddress.setText(helper.getRestaurantAddress(c));  
    restaurantTel.setText(helper.getRestaurantTel(c));  
  
    if (helper.getRestaurantType(c).equals("Chinese")) {  
        restaurantTypes.check(R.id.chinese);  
    } else if (helper.getRestaurantType(c).equals("Western")) {  
        restaurantTypes.check(R.id.western);  
    } else if (helper.getRestaurantType(c).equals("Indian")) {  
        restaurantTypes.check(R.id.indian);  
    } else if (helper.getRestaurantType(c).equals("Indonesia")) {  
        restaurantTypes.check(R.id.indonesian);  
    } else if (helper.getRestaurantType(c).equals("Korean")) {  
        restaurantTypes.check(R.id.korean);  
    } else if (helper.getRestaurantType(c).equals("Japanese")) {  
        restaurantTypes.check(R.id.japanese);  
    } else {  
        restaurantTypes.check(R.id.thai);  
    }  
}
```

39. When the restaurant record has been read and displayed in ‘Details’ form, what will happen if user clicks on the “Save” button?

To avoid adding the same restaurant record to the *restaurants\_table* in *restaurants* SQLite database, we can make use of the same checking method as Step 38, i.e. if the “*restaurantID*” is null, add a new record to the *restaurants\_table*. Otherwise, update the existing record in the *restaurants\_table*.

```
if (restaurantID == null) {  
    helper.insert(nameStr, addrStr, telStr, restType);  
} else {  
    helper.update(restaurantID, nameStr, addrStr, telStr, restType);  
}
```

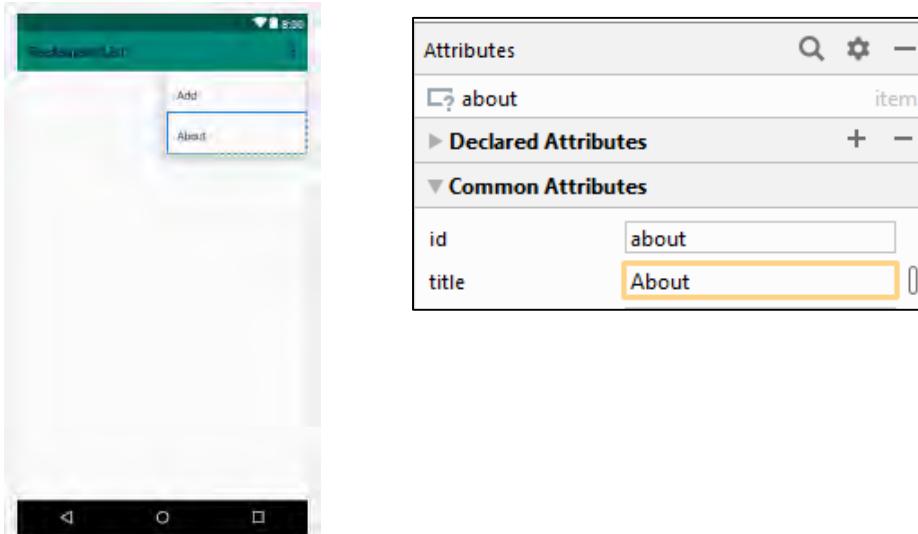
40. Run the Lab4 project. Test reading a record from **restaurants\_table**. Change some data when record is loaded in the ‘Details’ form and “save”. No new record should be added to the ‘List’

41. Show the result to your lecturer when this section has been completed successfully

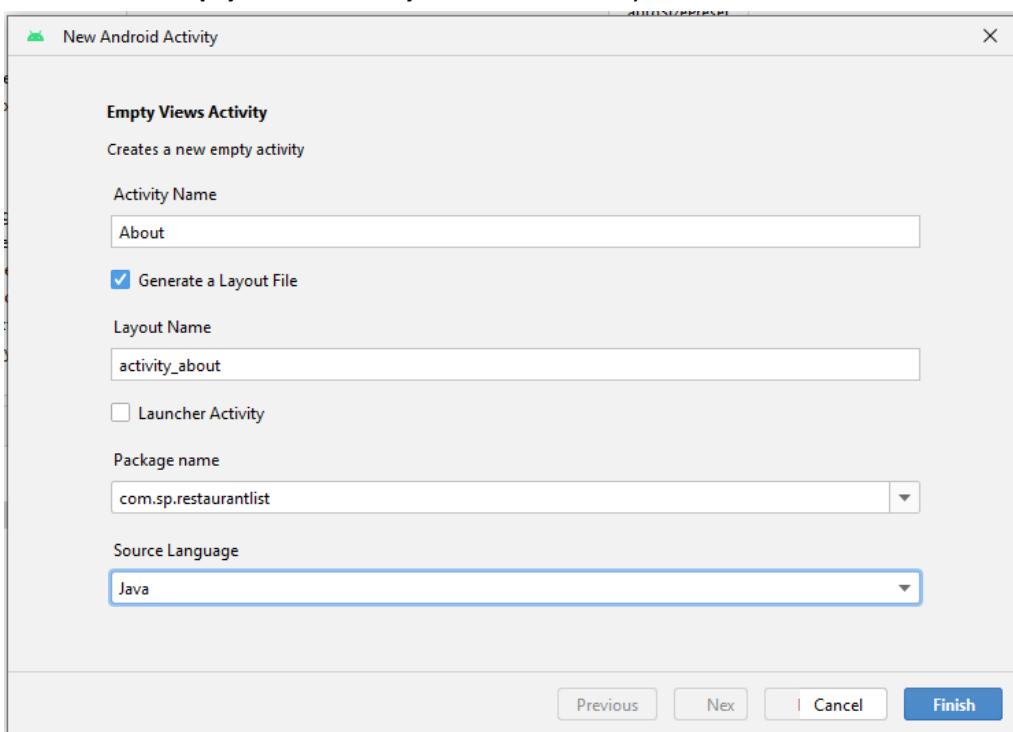
### Part III – “About” Activity

This is an exercise that you need to complete and show the result to your lecturer.

1. Add new menu item “About”. Skip this step if your app already has an “About” menu item.



2. Create a new **Empty Views Activity** – “About” with layout “about”.

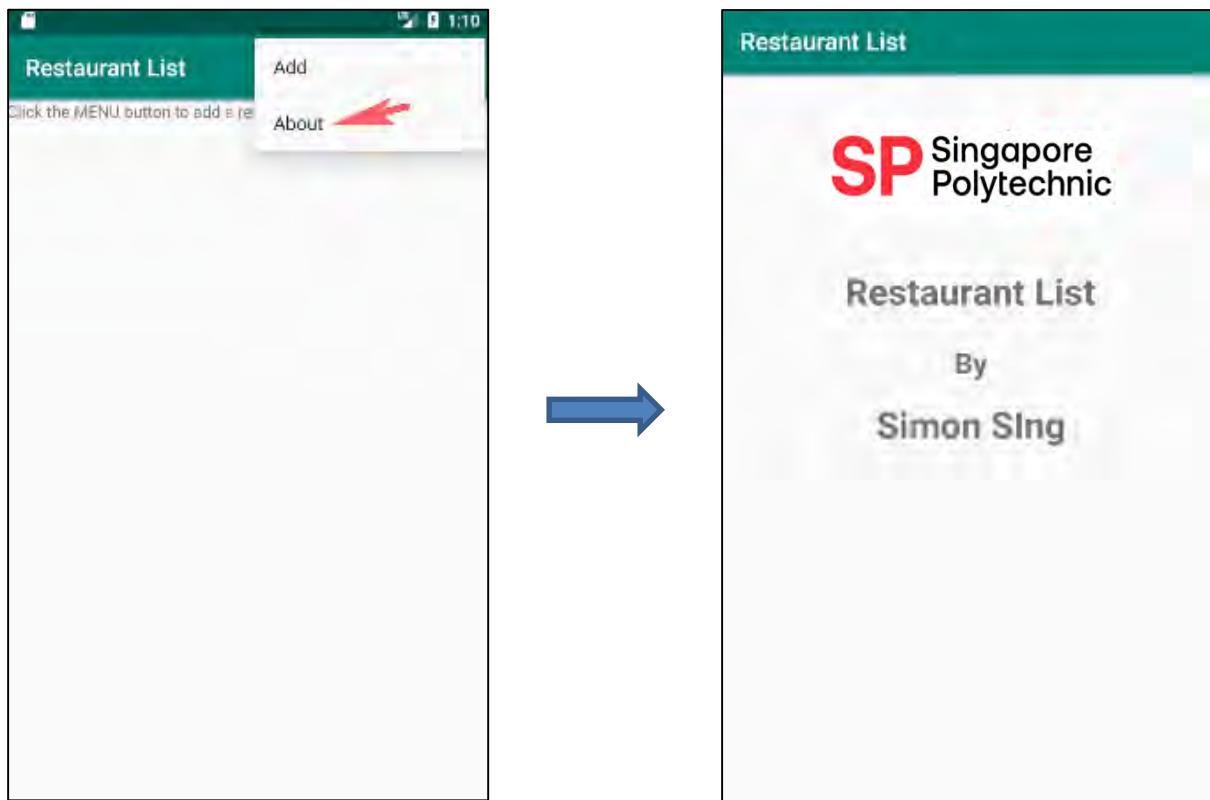


3. Design the “about” layout as follows. You may use your own image. Note that your **image filename** must be in **small letters with no space** e.g. “splogo.png” and **NOT** “SP Logo.png”.



4. Program the option selected event for the “About” menu item to launch the “About” activity when it is selected. Hint: refer to step 26.

**5. Show the result to your lecturer when this section has been completed successfully**

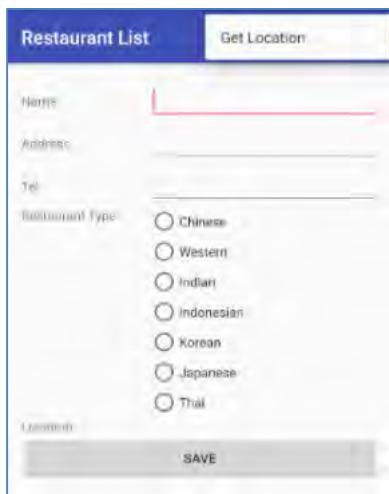


**-END-**

## Practical 5: GPS Location & Map View

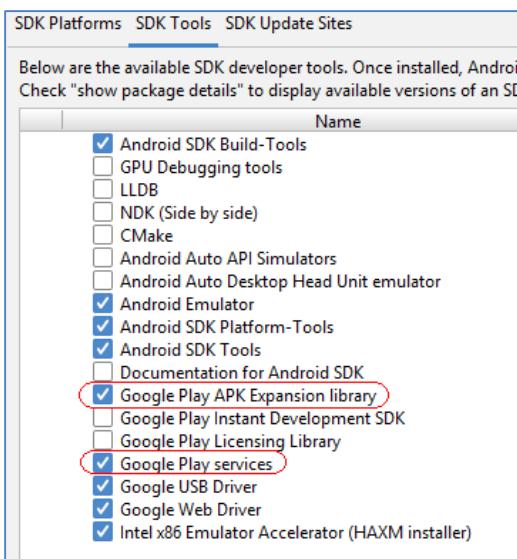
In this session, you will learn how to get a GPS coordinate for a restaurant and display the location on Google map with a marker

### Part I – Getting GPS Location



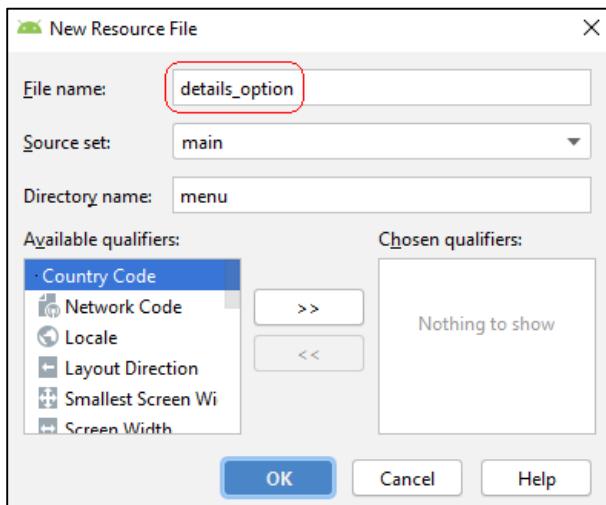
The screenshot shows a user interface for adding a new restaurant. At the top, there's a header bar with 'Restaurant List' and a 'Get Location' button. Below this is a form with fields for 'Name' (containing 'L'), 'Address', 'Tel', and 'Institutional Type'. The 'Institutional Type' section includes radio buttons for Chinese, Western, Indian, Indonesian, Korean, Japanese, and Thai. At the bottom is a 'Common' section with a 'SAVE' button.

1. It might be good if the Restaurant location can be shown on a map. In this exercise, the GPS coordinates of the restaurant will be saved in the database for later use. In order to incorporate Google Map into Restaurant List app, the following SDK Tools will be needed. At top menu bar, **select Tools > Android > SDK Manager** and check if these tools are installed or updated.



2. Create a new project with **No Activity** and the following information
  - **Name:** Restaurant List
  - **Package name:** com.sp.restaurantlist
  - **Save location:** C:\MAD\AndroidStudioProjects\Lab5a or D:\MAD\AndroidStudioProjects\Lab5a
  - **Language:** Java
  - **Minimum SDK: API 27: Android 8.1 (Oreo)**
  - **Source Language:** Java
  - **Build configuration language:** Groovy DSL (build.gradle)

3. Open your **Windows File Explorer** and navigate to your **Android Studio** workspace where all your projects are created.
4. Copy **AndroidManifest.xml** file, **java** and **res** folders from **Lab4\app\src\main** project folder and paste into **Lab5a\app\src\main** folder to overwrite the existing files and folders
5. At Android Studio Project Pane, right click on **res > menu** folder, select **New > Menu resource file** to create a new menu named **details\_option**

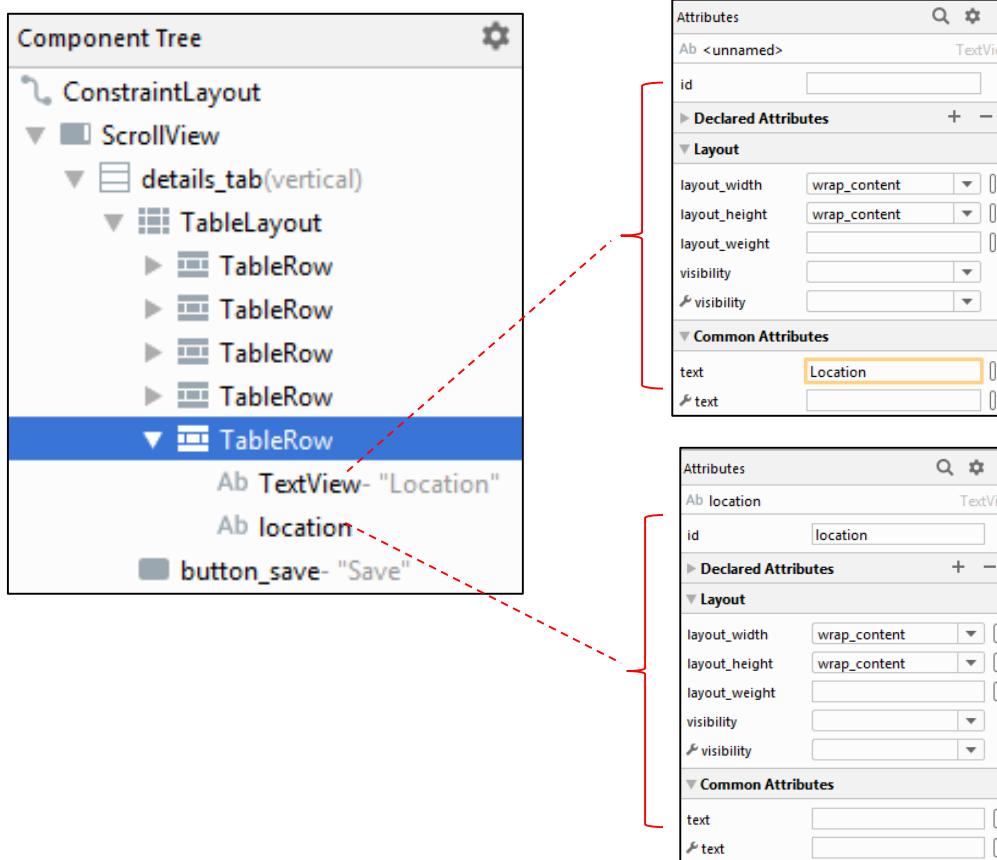


6. At Design Editor, drag and drop a Menu Item widget into **menu** widget under Component Tree. Update the item's attributes 'id' and 'title' with the values shown as follows

| Attribute | Value              |
|-----------|--------------------|
| id        | get_location       |
| title     | Get Location       |
| icon      | [Icon Placeholder] |

7. With an option 'Get Location' added, the next widget you need is to show the GPS coordinates on the UI display.

8. Open the **detail\_form** layout from **res > layout**. Drag and drop one **TableRow** and two **TextView** widgets into the **TableLayout** widget as shown. Update the attributes of the two **TextView** widgets with the following content.



9. Due to the additional GPS location, these information (latitude and longitude) will need to be saved as part of the record into the data model of **restaurants\_table** in **RestaurantHelper**.
10. Open the **RestaurantHelper.java** to update the **restaurants\_table** with two extra fields (latitude and longitude) to store the Restaurant Location information and the location processing methods in the helper. Make the changes as per the highlighted codes.

```

1 package com.sp.restaurantlist;
2
3 import android.content.ContentValues;
4 import android.content.Context;
5 import android.database.Cursor;
6 import android.database.sqlite.SQLiteDatabase;
7 import android.database.sqlite.SQLiteOpenHelper;
8
9 public class RestaurantHelper extends SQLiteOpenHelper {
10     private static final String DATABASE_NAME = "restaurantlist.db";
11     private static final int SCHEMA_VERSION = 1;
12
13     public RestaurantHelper(Context context) {
14         super(context, DATABASE_NAME, null, SCHEMA_VERSION);
15     }
16
17     @Override
18     public void onCreate(SQLiteDatabase db) {
19         // Will be called once when the database is not created
20         db.execSQL("CREATE TABLE restaurants_table ( _id INTEGER PRIMARY KEY AUTOINCREMENT, " +
21                 "restaurantName TEXT, restaurantAddress TEXT, restaurantTel TEXT, " +
22                 "restaurantType TEXT, lat REAL, lon REAL);");
23     }
24 }
```

Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

```
25  @Override
26  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
27      // Will not be called until SCHEMA_VERSION increases
28      // Here we can upgrade the database e.g. add more tables
29  }
30
31  /* Read all records from restaurants_table */
32  public Cursor getAll() {
33      return (getReadableDatabase().rawQuery(
34          "SELECT _id, restaurantName, restaurantAddress, restaurantTel, "
35          + "restaurantType, lat, lon FROM restaurants_table ORDER BY restaurantName", null));
36  }
37
38  /* Read a particular record from restaurants_table with id provided */
39  public Cursor getById(String id) {
40      String[] args = {id};
41
42      return (getReadableDatabase().rawQuery(
43          "SELECT _id, restaurantName, restaurantAddress, restaurantTel, "
44          + "restaurantType, lat, lon FROM restaurants_table WHERE _ID=?", args));
45  }
46
47  /* Write a record into restaurants_table */
48  public void insert(String restaurantName, String restaurantAddress, String restaurantTel,
49                      String restaurantType, double lat, double lon) {
50      ContentValues cv = new ContentValues();
51
52      cv.put("restaurantName", restaurantName);
53      cv.put("restaurantAddress", restaurantAddress);
54      cv.put("restaurantTel", restaurantTel);
55      cv.put("restaurantType", restaurantType);
56      cv.put("lat", lat);
57      cv.put("lon", lon);
58
59      getWritableDatabase().insert("restaurants_table", "restaurantName", cv);
60  }
61
62  /* Update a particular record in restaurants_table with id provided */
63  public void update(String id, String restaurantName, String restaurantAddress,
64                     String restaurantTel, String restaurantType, double lat, double lon) {
65      ContentValues cv = new ContentValues();
66      String[] args = {id};
67      cv.put("restaurantName", restaurantName);
68      cv.put("restaurantAddress", restaurantAddress);
69      cv.put("restaurantTel", restaurantTel);
70      cv.put("restaurantType", restaurantType);
71      cv.put("lat", lat);
72      cv.put("lon", lon);
73
74      getWritableDatabase().update("restaurants_table", cv, "_ID=?", args);
75  }
76
77  /* Read a record id value from restaurants_table */
78  public String getID(Cursor c) { return (c.getString(0)); }
79
80  @ + public String getRestaurantName(Cursor c) { return (c.getString(1)); }
81
82  @ + public String getRestaurantAddress(Cursor c) { return (c.getString(2)); }
83
84  @ + public String getRestaurantTel(Cursor c) { return (c.getString(3)); }
85
86  @ + public String getRestaurantType(Cursor c) { return (c.getString(4)); }
87
88  @ + public double getLatitude(Cursor c) { return (c.getDouble(5)); }
89
90  @ + public double getLongitude(Cursor c) { return (c.getDouble(6)); }
```

11. You have so far updated the Detail Form UI with an extra **TextView** widget for displaying latitude & longitude information, and update the SQLite Database Helper by introducing two extra fields ("lat" and "lon") to allow you to save the location latitude and longitude as part of a restaurant record in *restaurants\_table*.
12. The next step is to create a Service to tell the phone to get the GPS information in the background.  
**GPSTracker** is a subclass of **Service** created to access phone GPS and returns GPS coordinates values whenever *getLatitude()* or *getLongitude()* method is called.

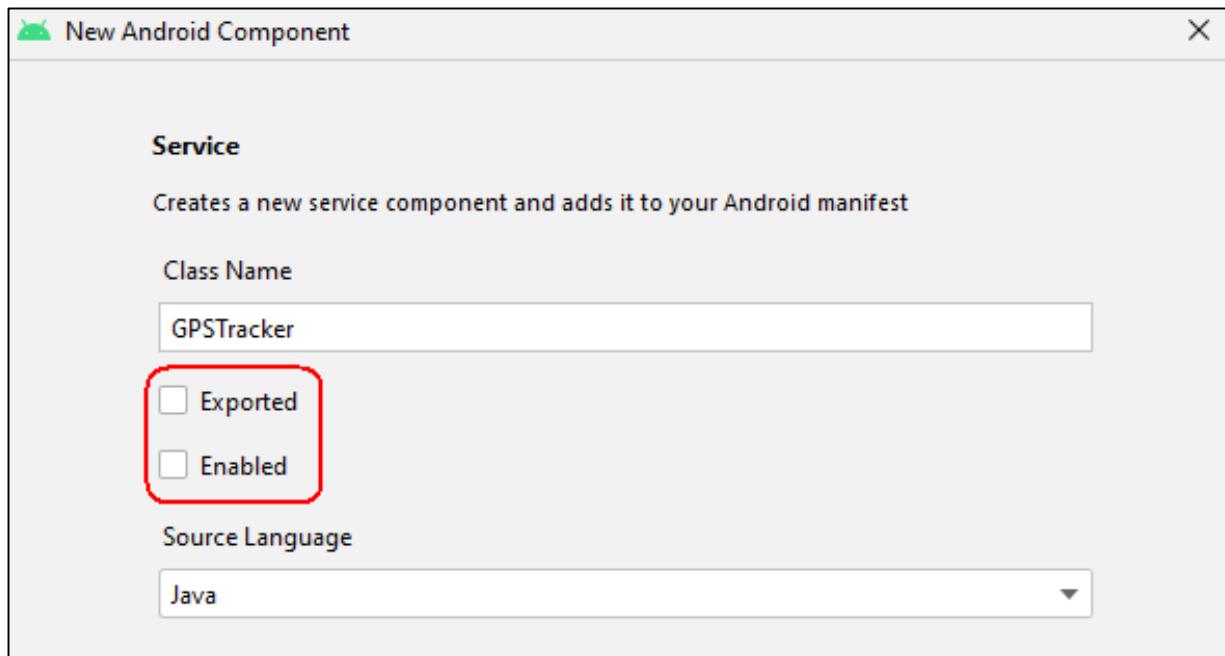
13. **\*IMPORTANT**

Due to security feature of Android, when an app requires to access phone GPS, **Permissions** must be set in the *Android Manifest* file. Open and edit *AndroidManifest.xml* as shown.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...

```

14. To create the **GPSTracker** service, right click on the **java > com.sp.restaurantlist** folder and select **New > Service > Service**. Enter the file name as 'GPSTracker'. Uncheck both **Exported** and **Enabled** options.



15. Update **GPSTracker.java** with the following content

```
1 package com.sp.restaurantlist;
2
3 import android.Manifest;
4 import android.app.Activity;
5 import android.app.Service;
6 import android.content.Context;
7 import android.content.Intent;
8 import android.content.pm.PackageManager;
9 import android.location.Location;
10 import android.location.LocationListener;
11 import android.location.LocationManager;
12 import android.os.Bundle;
13 import android.os.IBinder;
14 import androidx.annotation.NonNull;
15 import androidx.core.app.ActivityCompat;
16
17 public class GPSTracker extends Service implements LocationListener, ActivityCompat.OnRequestPermissionsResultCallback {
18     private static final int GPS_PERMISSION_REQUEST_CODE = 1001;
19     private Context mContext = null;
20     boolean canGetLocation = false; // flag for GPS status
21
22     Location location; // location
23     double latitude; // latitude
24     double longitude; // longitude
25     // Declaring a Location Manager
26     protected LocationManager locationManager;
27
28     public GPSTracker() {
29         getLocation();
30     }
31
32     public GPSTracker(Context context) {
33         this.mContext = context;
34         getLocation();
35     }
36
37     private void getLocation() {
38         this.canGetLocation = false;
39         int permissionState1 = ActivityCompat.checkSelfPermission(mContext,
40             Manifest.permission.ACCESS_FINE_LOCATION);
41         int permissionState2 = ActivityCompat.checkSelfPermission(mContext,
42             Manifest.permission.ACCESS_COARSE_LOCATION);
43         if (permissionState1 == PackageManager.PERMISSION_GRANTED &&
44             permissionState2 == PackageManager.PERMISSION_GRANTED) {
45             // Permission granted, get GPS location
46             locationManager = (LocationManager) mContext.getSystemService(Context.LOCATION_SERVICE);
47             location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
48             if (location != null) {
49                 latitude = location.getLatitude();
50                 longitude = location.getLongitude();
51             }
52             this.canGetLocation = true;
53         } else {
54             ActivityCompat.requestPermissions((Activity)mContext, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
55   GPS_PERMISSION_REQUEST_CODE);
56         }
57     }
58 }
```

(If LocationManager keeps reading 0,0 for latitude and longitude, try NETWORK\_PROVIDER instead of GPS\_PROVIDER)

```
58
59
60     @Override
61     public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
62         if (requestCode == GPS_PERMISSION_REQUEST_CODE) {
63             if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
64                 // Permission granted, continue with the location access
65                 getLocation();
66             } else {
67                 // Permission denied, show a message or take appropriate action
68                 ActivityCompat.requestPermissions((Activity)mContext, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
69   GPS_PERMISSION_REQUEST_CODE);
70             }
71         }
72     }
73
74     // Function to get latitude
75     public double getLatitude() {
76         if (location != null) {
77             latitude = location.getLatitude();
78         }
79         return latitude;
80     }
81
82     // Function to get longitude
83     public double getLongitude() {
84         if (location != null) {
85             longitude = location.getLongitude();
86         }
87         return longitude;
88     }
89
90     public boolean canGetLocation() {
91         getLocation();
92         return canGetLocation;
93     }
94
95     /**
96      * Stop using GPS listener
97      * Calling this function will stop using GPS in your app
98      */
99
100    public void stopUsingGPS() {
101        if (locationManager != null) {
102            locationManager.removeUpdates(GPSTracker.this);
103        }
104    }
105}
```

```
102  
103  
104     @Override  
105     public void onLocationChanged(Location location) {  
106         getLocation();  
107     }  
108  
109     @Override  
110     public void onProviderDisabled(String provider) {  
111     }  
112  
113     @Override  
114     public void onProviderEnabled(String provider) {  
115     }  
116  
117     @Override  
118     public void onStatusChanged(String provider, int status, Bundle extras) {  
119     }  
120  
121     @Override  
122     public IBinder onBind(Intent intent) {  
123         // TODO: Return the communication channel to the service.  
124         throw new UnsupportedOperationException("Not yet implemented");  
125     }  
126 }
```

16. Whenever the “**Get Location**” menu item is selected, the GPSTracker (**Service** subclass) will get GPS location information and display into the new **TextView** introduced in the ‘Details’ form UI view.
17. Open the *DetailForm.java* file and update with the **highlighted** content.

```
1 package com.sp.restaurantlist;
2
3 import android.database.Cursor;
4 import android.os.Bundle;
5 import android.view.Menu;
6 import android.view.MenuInflater;
7 import android.view.MenuItem;
8 import android.view.View;
9 import android.widget.Button;
10 import android.widget.EditText;
11 import android.widget.RadioGroup;
12 import android.widget.TextView;
13 import android.widget.Toast;
14 import androidx.appcompat.app.AppCompatActivity;
15
16 public class DetailForm extends AppCompatActivity {
17     private EditText restaurantName;
18     private EditText restaurantAddress;
19     private EditText restaurantTel;
20     private RadioGroup restaurantTypes;
21     private Button buttonSave;
22
23     private RestaurantHelper helper = null;
24     private String restaurantID = "";
25
26     private TextView location = null;
27     private GPSTracker gpsTracker;
28     private double latitude = 0.0d;
29     private double longitude = 0.0d;
30 }
```

Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

```
31
32     ↗
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54     ↗
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.detail_form);
    restaurantName = findViewById(R.id.restaurant_name);
    restaurantAddress = findViewById(R.id.restaurant_address);
    restaurantTel = findViewById(R.id.restaurant_tel);
    restaurantTypes = findViewById(R.id.restaurant_types);

    buttonSave = findViewById(R.id.button_save);
    buttonSave.setOnClickListener(onSave);
    helper = new RestaurantHelper(this);

    location = findViewById(R.id.location);
    gpsTracker = new GPSTracker(DetailForm.this);

    restaurantID = getIntent().getStringExtra("ID");
    if (restaurantID != null) {
        load();
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    helper.close();
    gpsTracker.stopUsingGPS();
}

private void load() {
    Cursor c = helper.getById(restaurantID);
    c.moveToFirst();
    restaurantName.setText(helper.getRestaurantName(c));
    restaurantAddress.setText(helper.getRestaurantAddress(c));
    restaurantTel.setText(helper.getRestaurantTel(c));

    if (helper.getRestaurantType(c).equals("Chinese")) {
        restaurantTypes.check(R.id.chinese);
    } else if (helper.getRestaurantType(c).equals("Western")) {
        restaurantTypes.check(R.id.western);
    } else if (helper.getRestaurantType(c).equals("Indian")) {
        restaurantTypes.check(R.id.indian);
    } else if (helper.getRestaurantType(c).equals("Indonesian")) {
        restaurantTypes.check(R.id.indonesian);
    } else if (helper.getRestaurantType(c).equals("Korean")) {
        restaurantTypes.check(R.id.korean);
    } else if (helper.getRestaurantType(c).equals("Japanese")) {
        restaurantTypes.check(R.id.japanese);
    } else {
        restaurantTypes.check(R.id.thai);
    }

    latitude = helper.getLatitude(c);
    longitude = helper.getLongitude(c);
    location.setText(String.valueOf(latitude) + ", " + String.valueOf(longitude));
}
```

Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

```
B7
B8
B9  ↗
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
B24
B25
B26
B27
B28
B29
B30
B31
B32
B33
B34
B35
B36
B37
B38
B39
B40
B41
B42
B43
B44
B45
B46
B47
B48
B49
B50
B51
B52
B53
B54
B55
B56
B57
B58 ↗ @
B59
B60
B61
B62
B63
B64
B65
B66
B67
B68
B69
B70
B71
B72
B73
B74
B75
B76
B77
B78
B79
B80
B81
B82
B83
B84
B85
B86
B87
B88
B89
B90
B91
B92
B93
B94
B95
B96
B97
B98
B99
B100
B101
B102
B103
B104
B105
B106
B107
B108
B109
B110
B111
B112 ↗ ↑
B113
B114
B115
B116
B117
B118
B119
B120
B121
B122
B123
B124
B125
B126
B127
B128
B129
B130
B131
B132
B133
B134
B135
B136
B137
B138
B139
B140
B141
B142
B143
B144
B145
B146
B147
B148
B149
B150
B151
B152
B153

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    new MenuInflater(this).inflate(R.menu.details_option, menu);
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.get_location) {
        if (gpsTracker.canGetLocation()) {
            latitude = gpsTracker.getLatitude();
            longitude = gpsTracker.getLongitude();
            location.setText(String.valueOf(latitude) + ", " + String.valueOf(longitude));
            // \n is for new line
            Toast.makeText(getApplicationContext(), "Your Location is - \nLat: " + latitude
                + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
        }
        return (true);
    }
    return super.onOptionsItemSelected(item);
}

View.OnClickListener onSave = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // To read date from EditText
        String nameStr = restaurantName.getText().toString();
        String addrStr = restaurantAddress.getText().toString();
        String telStr = restaurantTel.getText().toString();
        String restType = "";
        //To read selection of restaurantTypes RadioGroup
        int radioID = restaurantTypes.getCheckedRadioButtonId();
        if (radioID == R.id.chinese ) {
            restType = "Chinese";
        } else
        if (radioID == R.id.western ) {
            restType = "Western";
        } else
        if (radioID == R.id.indian ) {
            restType = "Indian";
        } else
        if (radioID == R.id.indonesian ) {
            restType = "Indonesian";
        } else
        if (radioID == R.id.korean) {
            restType = "Korean";
        } else
        if (radioID == R.id.japanese) {
            restType = "Japanese";
        } else
        if (radioID == R.id.thai) {
            restType = "Thai";
        }

        if (restaurantID == null) {
            helper.insert(nameStr, addrStr, telStr, restType, latitude, longitude);
        } else {
            helper.update(restaurantID, nameStr, addrStr, telStr, restType, latitude, longitude);
        }

        //To close current Activity class and exit
        finish();
    }
};
```

18. Check that AndroidManifest.xml has the setting as shown.

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

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Restaurant List"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.RestaurantList"
        tools:targetApi="31">
        <service
            android:name=".GPSTracker"
            android:enabled="false"
            android:exported="false"></service>
        <activity
            android:name=".About"
            android:exported="false" />
        <activity
            android:name=".DetailForm"
            android:exported="false" />
        <activity
            android:name=".RestaurantList"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

19. To test *Lab5a* project, you will need to make use of actual mobile phone.

[**Note** Due to the change of the *restaurants\_table* data structure, you need to **uninstall the Restaurant List app in the phone before running**. Otherwise, you will encounter fatal error]

20. At the '**Detail**' form, click the **MENU** button and select "**Get Location**" option.

If you have not enable the "**Location Permission**", you will be prompted to do so.

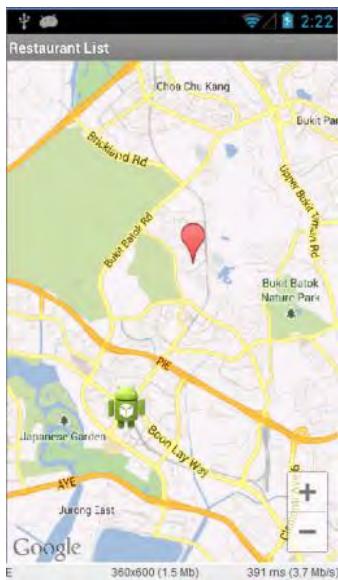
Similarly, if you have not turn on the "**Location**" (GPS) in your settings, you will be prompted to do so.

A **Toast** message of the current location latitude and longitude will be displayed.

[**Note** Make sure that the phone GPS is enabled and there is at least WiFi connection]

21. Show the result to your lecturer when you have completed this section

## Part II – Using Android Google Map

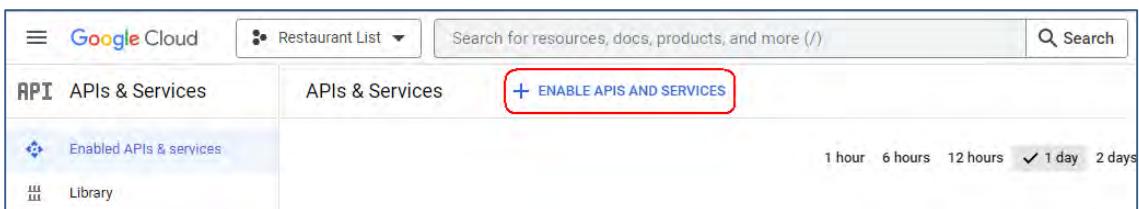


22. With the GPS coordinates for the restaurant, it might be useful to show the location on Google map.
  23. Create a new project with **No Activity** and the following information
    - **Name:** Restaurant List
    - **Package name:** com.sp.restaurantlist
    - **Save location:** C:\MAD\AndroidStudioProjects\Lab5b or D:\MAD\AndroidStudioProjects\Lab5b
    - **Language:** Java
    - **Minimum SDK:** API 27: Android 8.1 (Oreo)
    - **Source Language:** Java
    - **Build configuration language:** Groovy DSL (build.gradle)
  24. Open your **Windows File Explorer** and navigate to your **Android Studio** workspace where all your projects are created.
  25. Copy *AndroidManifest.xml* file, **java** and **res** folders from **Lab5a\app\src\main** project folder and paste into **Lab5b\app\src\main** folder to overwrite the existing files and folders.
- ### Creating Android Google Map Key
26. To use the Android Google Map services, you will need to register for an API key at the following site  
<https://console.cloud.google.com/projectselector2/apis/dashboard>  
**[Note: You need a Gmail account]**
  27. After you have logged into your Gmail account, click on the “**CREATE**” to create a project with the name ‘Restaurant List’

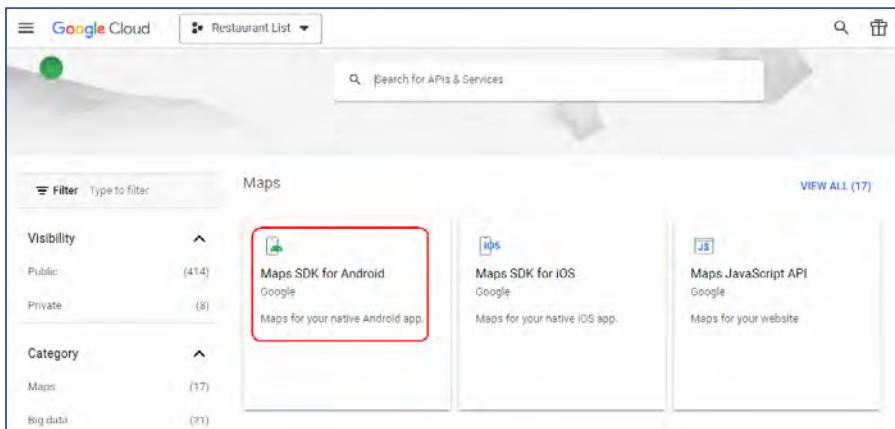
**Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering**

The screenshot shows the Google Cloud API Services dashboard. A modal window is open for creating a new project. The project name is set to 'Restaurant List'. The location is set to 'No organization'. There are 'CREATE' and 'CANCEL' buttons at the bottom. A red box highlights the 'CREATE PROJECT' button in the top right corner of the modal.

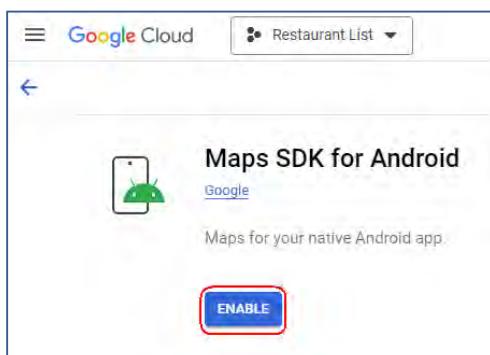
28. After creating 'Restaurant List' project, click "**ENABLE APIS AND SERVICES**".



29. Select **Maps SDK for Android** under **Maps** category.



30. Click on the **ENABLE** button to enable the **Map SDK for Android**.



31. Select the “**CREDENTIALS**”.

The screenshot shows the Google Cloud Platform interface for managing APIs & Services. The 'Credentials' tab is active. A prominent red box highlights the '+ CREATE CREDENTIALS' button. Below it, a message says 'Create credentials to access your enabled APIs.' and a warning about configuring the OAuth consent screen. The 'API Keys' section shows a table with no data.

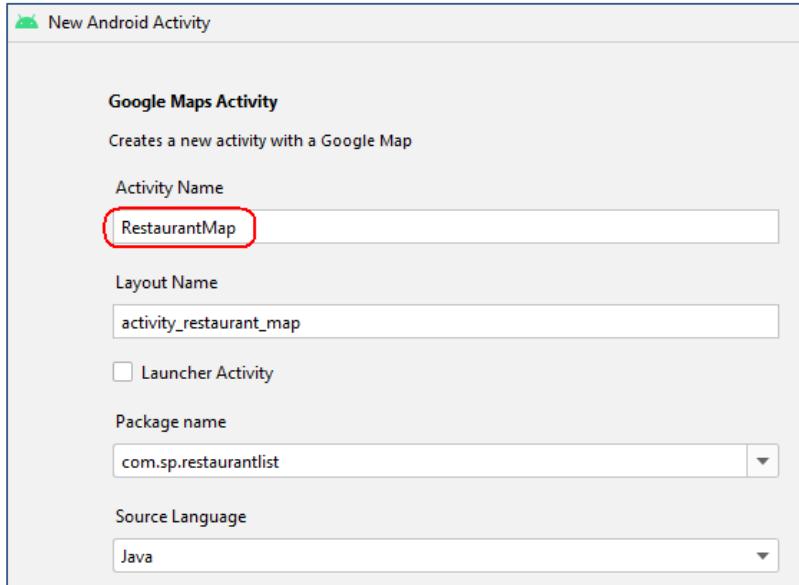
32. Click “**Create Credentials**”. Select “**API key**”. An API key will be created. Copy the key.

The first screenshot shows the 'CREATE CREDENTIALS' page with the 'API key' option selected, which is highlighted with a red box. The second screenshot shows the 'API key created' confirmation page with the generated API key displayed in a text input field.

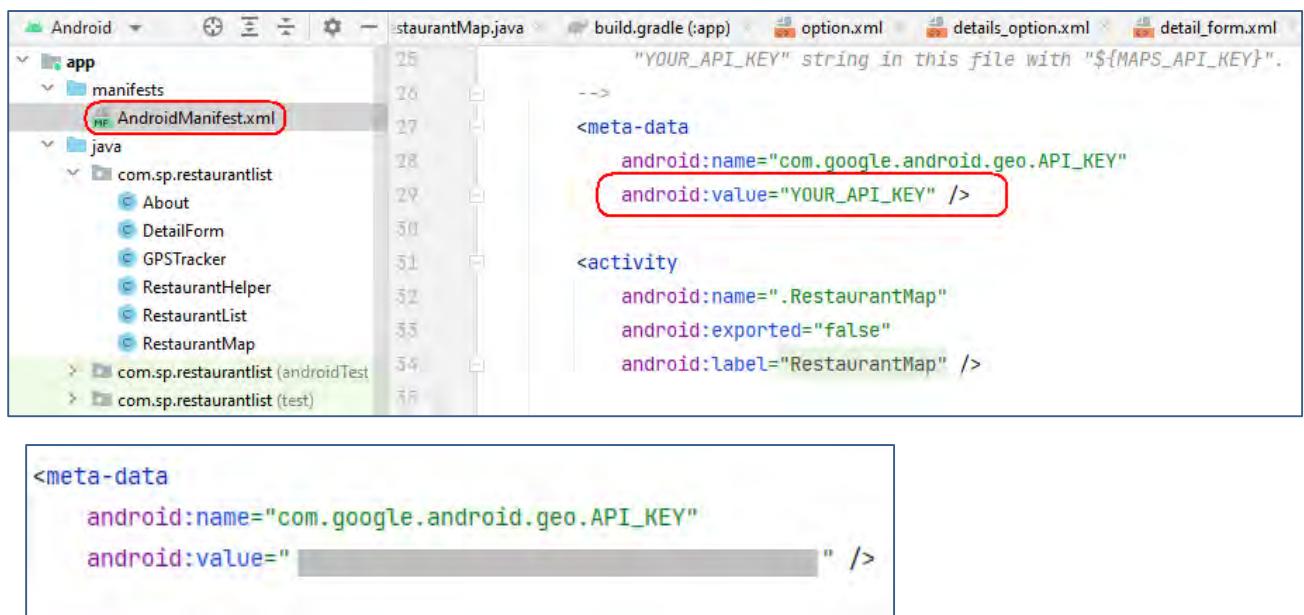
33. At Android Studio, right click on **com.sp.restaurant** folder. Select **New > Google > Google Maps Views**

The screenshot shows the Android Studio file structure. A right-click context menu is open over the 'com.sp.restaurant' folder. The path 'New > Google > Google Maps Views' is highlighted with a blue box. Other options in the menu include 'New Class', 'Kotlin Class/File', 'Android Resource File', etc.

34. Enter the activity's name as **RestaurantMap**.



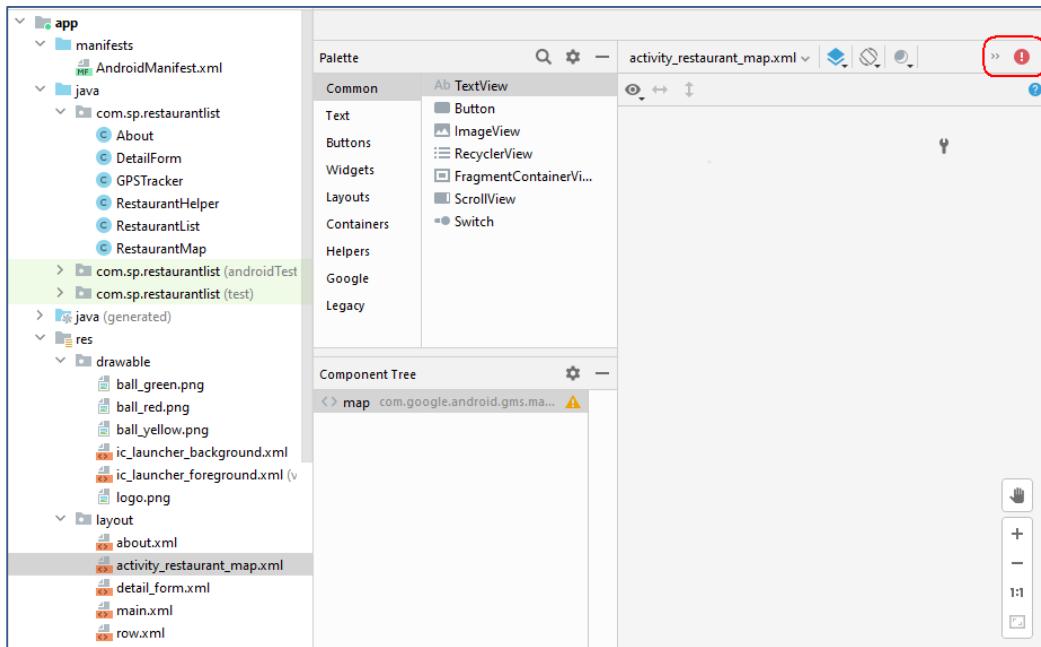
35. Open the **AndroidManifest.xml**. Copy the generated Android API Key (**STEP 34**) and overwrite **YOUR\_API\_KEY**.



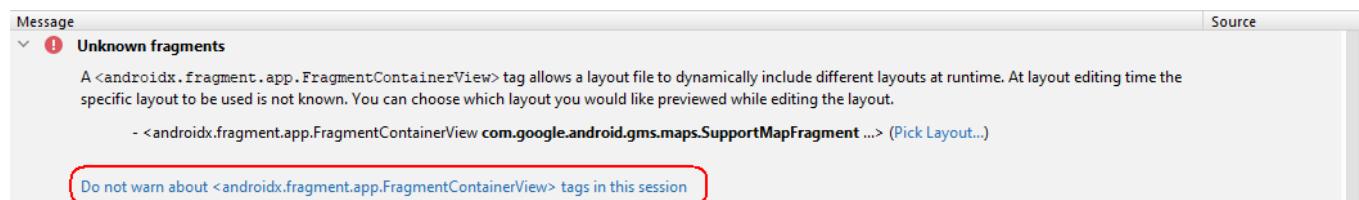
36. At **res/layout** folder, double click on **activity\_restaurant\_map.xml** to open the UI view file for Google Map. In the “Component Tree” select “map”.

**Official (Closed), Non-Sensitive  
SINGAPORE POLYTECHNIC  
School of Electrical & Electronic Engineering**

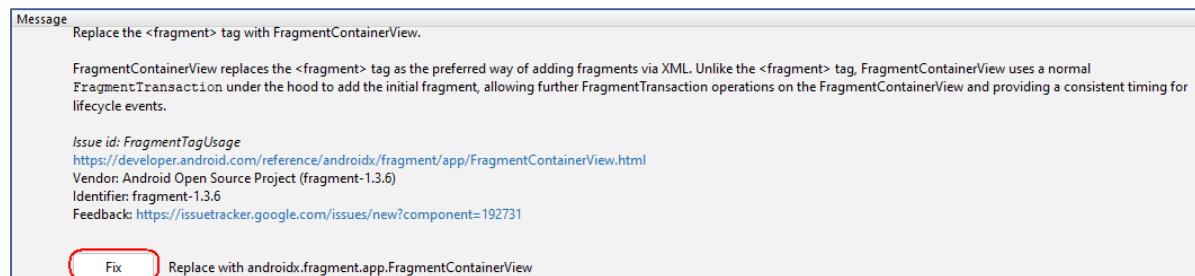
37. Click the red error icon on the top right corner.



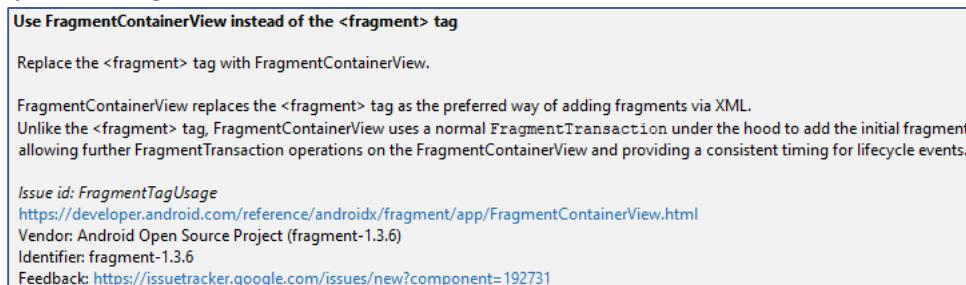
Select “Do not warn about <androidx.fragment.app.FragmentContainerView> tags in this session.”



Click the “Fix” button (if available) for the “Use FragmentContainerView instead of the <fragment> tag” warning message.

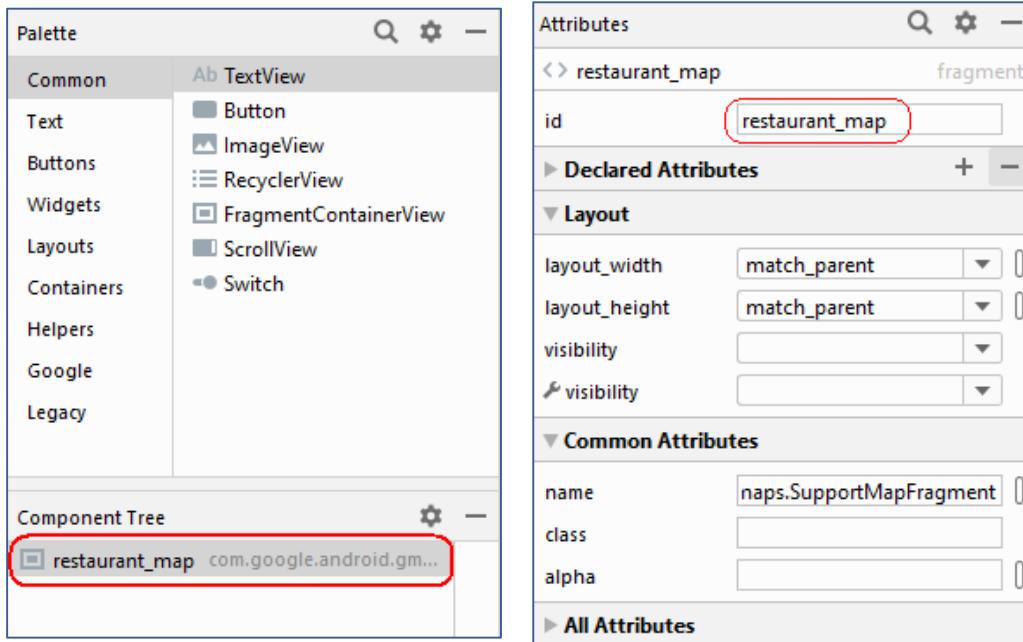


Or just warning



Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

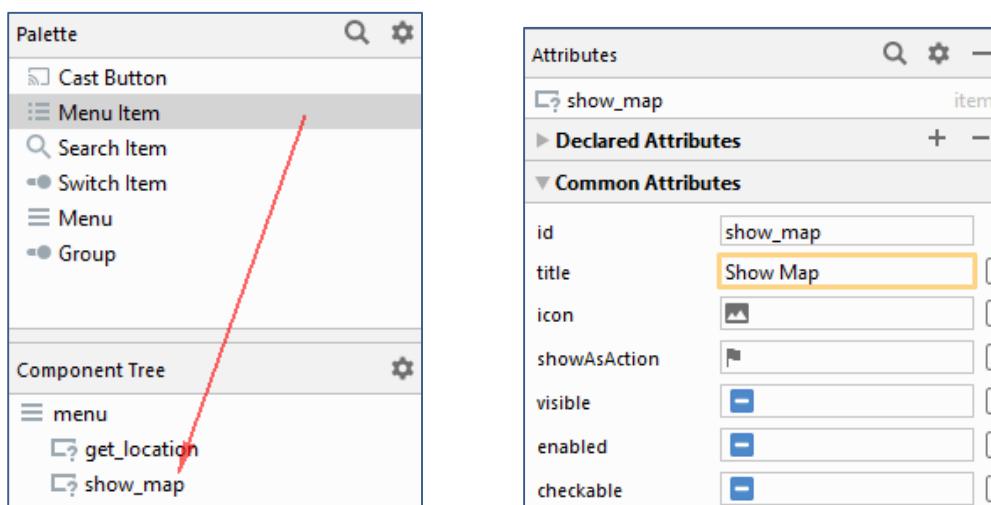
38. On the Attribute tab, update the map ID to “restaurant\_map”.



39. Check that your “activity\_restaurant\_map.xml” code is as below. Edit accordingly if different.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/restaurant_map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RestaurantMap" />
```

40. Download *marker.png* and *marker\_me.png* image icons from Brightspace under **Learning Resources > Practical Session Resources > “Image icons might be used in Labs”** folder and save to **res/drawable** folder.  
 41. Open the *details\_option.xml* file in **res/menu** folder to add in the new **Show Map** option item to the MENU. Update the item **id** to “**show\_map**” and **title** to “**Show Map**” for widget properties as shown



42. Update new Java class ***RestaurantMap.java*** with the highlighted contents.

```
1 package com.sp.restaurantlist;
2
3 import androidx.fragment.app.FragmentActivity;
4 import android.os.Bundle;
5 import com.google.android.gms.maps.CameraUpdateFactory;
6 import com.google.android.gms.maps.GoogleMap;
7 import com.google.android.gms.maps.OnMapReadyCallback;
8 import com.google.android.gms.maps.SupportMapFragment;
9 import com.google.android.gms.maps.model.BitmapDescriptorFactory;
10 import com.google.android.gms.maps.model.LatLng;
11 import com.google.android.gms.maps.model.Marker;
12 import com.google.android.gms.maps.model.MarkerOptions;
13
14 public class RestaurantMap extends FragmentActivity implements OnMapReadyCallback {
15
16     private GoogleMap mMap;
17     private double lat;
18     private double lon;
19     private String restaurantName;
20     private double myLat;
21     private double myLon;
22     private LatLng RESTAURANT;
23     private LatLng ME;
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.activity_restaurant_map);
29
30         lat = getIntent().getDoubleExtra("LATITUDE", 0);
31         lon = getIntent().getDoubleExtra("LONGITUDE", 0);
32         restaurantName = getIntent().getStringExtra("NAME");
33         myLat = getIntent().getDoubleExtra("MYLATITUDE", 0);
34         myLon = getIntent().getDoubleExtra("MYLONGITUDE", 0);
35
36         // Obtain the SupportMapFragment and get notified when the map is ready to be used.
37         SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
38             .findFragmentById(R.id.restaurant_map);
39         mapFragment.getMapAsync(this);
40     }
}
```

```

41
42
43     * Manipulates the map once available.
44     * This callback is triggered when the map is ready to be used.
45     * This is where we can add markers or lines, add listeners or move the camera. In this case,
46     * we just add a marker near Sydney, Australia.
47     * If Google Play services is not installed on the device, the user will be prompted to install
48     * it inside the SupportMapFragment. This method will only be triggered once the user has
49     * installed Google Play services and returned to the app.
50
51     */
52
53     @Override
54     public void onMapReady(GoogleMap googleMap) {
55         mMap = googleMap;
56
57         RESTAURANT = new LatLng(lat, lon);
58         ME = new LatLng(myLat, myLon);
59
60         Marker restaurant = mMap.addMarker(new MarkerOptions().position(RESTAURANT).title(restaurantName));
61         Marker me = mMap.addMarker(new MarkerOptions().position(ME).title("ME")
62             .snippet("My location")
63             .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_me)));
64
65         // Move the camera instantly to restaurant with a zoom of 15.
66         mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(RESTAURANT, 15));
67     }

```

43. Open the *DetailForm.java* file and update the code for **Show Map** menu item with the highlighted contents.

- a) Add two new variables “myLatitude” and “myLongitude”.

```

1  package com.sp.restaurantlist;
2
3  import android.content.Intent;
4  import android.database.Cursor;
5  import android.os.Bundle;
6  import android.view.Menu;
7  import android.view.MenuInflater;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.widget.Button;
11 import android.widget.EditText;
12 import android.widget.RadioGroup;
13 import android.widget.TextView;
14 import android.widget.Toast;
15 import androidx.appcompat.app.AppCompatActivity;
16 public class DetailForm extends AppCompatActivity {
17     private EditText restaurantName;
18     private EditText restaurantAddress;
19     private EditText restaurantTel;
20     private RadioGroup restaurantTypes;
21     private Button buttonSave;
22     private RestaurantHelper helper = null;
23     private String restaurantID = "";
24     private TextView location = null;
25     private GPSTracker gpsTracker;
26     private double latitude = 0.0d;
27     private double longitude = 0.0d;
28     private double myLatitude = 0.0d;
29     private double myLongitude = 0.0d;

```

- b) Update “onOptionsItemSelected” event handler with the highlighted contents.



```
97     @Override
98     public boolean onOptionsItemSelected(MenuItem item) {
99         if (item.getItemId() == R.id.get_location) {
100             if (gpsTracker.canGetLocation()) {
101                 latitude = gpsTracker.getLatitude();
102                 longitude = gpsTracker.getLongitude();
103                 location.setText(String.valueOf(latitude) + ", " + String.valueOf(longitude));
104                 // \n is for new line
105                 Toast.makeText(getApplicationContext(), "Your Location is - \nLat: " + latitude
106                             + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
107             }
108             return (true);
109         } else if (item.getItemId() == R.id.show_map) {
110             //Get my current location
111             myLatitude = gpsTracker.getLatitude();
112             myLongitude = gpsTracker.getLongitude();
113
114             Intent intent = new Intent(this, RestaurantMap.class);
115             intent.putExtra("LATITUDE", latitude);
116             intent.putExtra("LONGITUDE", longitude);
117             intent.putExtra("MYLATITUDE", myLatitude);
118             intent.putExtra("MYLONGITUDE", myLongitude);
119             intent.putExtra("NAME", restaurantName.getText().toString());
120             startActivity(intent);
121             return (true);
122         }
123     }
124 }
```

44. To test this part of the program, it is good to use an actual Android phone. Under your Android device enable **USB debugging**. (The steps might vary from phone model to phone model.)

### Enable USB debugging on your Android phone

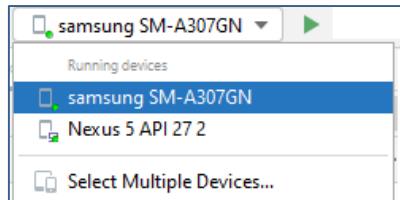
On Android 4.1 and lower, the Developer options screen is available by default. On Android 4.2 and higher, do the following:

1. Open the **Settings** app.
2. Select **System**.
3. Scroll to the bottom and select **About phone**.
4. Scroll to the bottom and tap **Build number** 7 times.
5. Return to the previous screen to find **Developer options** near the bottom.
6. Scroll down and enable **USB debugging**.

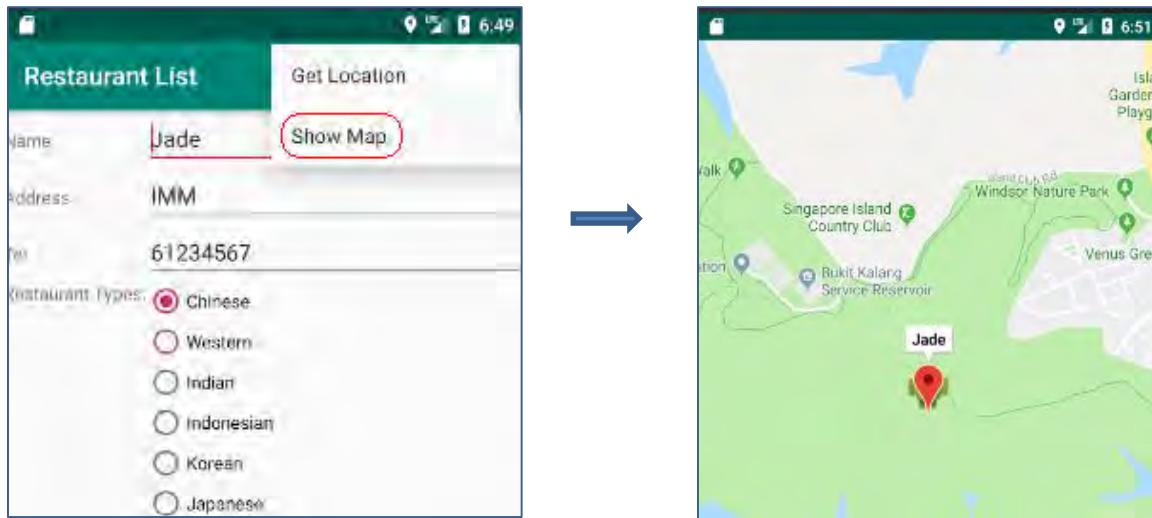
45. Plug the USB cable to your PC and make sure your device driver has successfully installed. Otherwise, **Android Studio** will not be able to detect your Android device.

Official (Closed), Non-Sensitive  
**SINGAPORE POLYTECHNIC**  
**School of Electrical & Electronic Engineering**

46. To test the program on the connected Android phone, select the connected device as shown.



47. Select the **Show Map** option from the **MENU**, the restaurant location will be shown on the map with two markers. If you click on the marker, a **Toast** will be shown with the restaurant name or your location  
[Note: Make sure you have Internet connection before testing]



**48. Show the result to your lecturer if you have completed this section**

**\*\*\* Extra\*\*\***

How to hardcode the GPS location of "ME" so that "ME" shows up as at Clementi Mall instead of overlapping with the restaurant location (i.e. SP GPS location)?

**-END-**

# ENGINEERING @ SP

**The School of Electrical & Electronic Engineering at Singapore Polytechnic offers the following full-time courses.**

## **1. Diploma in Aerospace Electronics (DASE)**

The Diploma in Aerospace Electronics course aims to provide students with skills and competencies in aerospace engineering avionics domains and emerging technologies in Infocomm-Technology to develop them to be a versatile engineer in both aerospace and adjacent industries.

## **2. Diploma in Computer Engineering (DCPE)**

This diploma aims to train technologists who can design, develop, setup and maintain computer systems; and develop software solutions. Students can choose to specialise in two areas of Computer Engineering & Infocomm Technology, which include Computer Applications, Smart City Technologies (IoT, Data Analytics), Cyber Security, and Cloud Systems.

## **3. Diploma in Electrical & Electronic Engineering (DEEE)**

This diploma offers a full range of modules in the electrical and electronic engineering spectrum. Students can choose one of the seven available specialisations (Biomedical, Communication, Microelectronics, Power, Rapid Transit Technology, Robotics & Control and Sustainable Energy) for their final year.

## **4. Diploma in Engineering with Business (DEB)**

The Diploma in Engineering with Business course aims to provide students with the requisite knowledge and skills in engineering principles, technologies, and business fundamentals, supported by a strong grounding in mathematics and communication skills, which is greatly valued in the rapidly changing industrial and commercial environment.

## **5. Common Engineering Program (DCEP)**

In Common Engineering Program, students will get a flavour of electrical, electronic and mechanical engineering in the first semester of their study. They will then choose one of the 7 engineering courses specially selected from the Schools of Electrical & Electronic Engineering and Mechanical & Aeronautical Engineering.

### School of Electrical & Electronic Engineering

More than  
**60 Years**  
of solid  
foundation

**8 Tech  
Hubs**

Unique  
**PTN**  
Scheme

**SP-NUS**  
**SP-SUTD**  
Accelerated  
Pathway  
Programmes

More than  
**35,000+**  
Alumni

# Electives offered by



SCHOOL OF

# Electrical & Electronic Engineering

All SP students, including EEE students are free to choose electives offered by ANY SP schools, subject to meeting the eligibility criteria.

Like all schools, School of Electrical and Electronic Engineering offers electives for:

- EEE students only and
- for all SP students

EEE students are required to complete 3 electives, starting from Year 2 to Year 3 (one elective per semester).

They may also choose to complete 4 or more electives in a particular category to receive a "Minor".

## Electives Choices for All SP students

| Mod Code | Module Title                                    |
|----------|-------------------------------------------------|
| EP0400   | Unmanned Aircraft Flying and Drone Technologies |
| EP0401   | Python Programming for IoT*                     |
| EP0402   | Fundamentals of IoT*                            |
| EP0403   | Creating an IoT Project*                        |
| EP0404   | AWS Academy Cloud Foundations                   |
| EP0405   | AWS Academy Cloud Architecting                  |
| EP0407   | Technology to Business                          |
| EP0408   | Cybersecurity Essentials                        |
| EP0409   | Low Code 5G & AIoT*                             |
| EP0410   | Fundamentals Of Electric Vehicle                |
| EP0411   | 5G & Next Generation Networks                   |
| EP0412   | Linux Essentials                                |

## Electives Choices for EEE students

| Mod Code | Module Title                                   |
|----------|------------------------------------------------|
| EM0400   | Commercial Pilot Theory                        |
| EM0401   | Autonomous Electric Vehicle Design             |
| EM0402   | Artificial Intelligence for Autonomous Vehicle |
| EM0403   | Autonomous Mobile Robots                       |
| EM0407   | Advanced Linux                                 |
| EM0408   | Linux System Administration                    |
| EM0409   | Rapid Transit System                           |
| EM0412   | Data Analytics                                 |
| EM0413   | Mobile App Development                         |
| EM0415   | Machine Learning & Artificial Intelligence     |
| EM0416   | Solar Photovoltaic System Design               |
| EM0418   | Integrated Building Energy Management System   |
| EM0427   | Commercial Pilot Theory 2                      |

### Certificate in Internet of Things (IoT)

- \* A certificate in IoT would be awarded if a student completes any 3 of the following modules: EP0401, EP0402, EP0403 and EP0409

### Minor in 5G & Artificial Intelligence of Things (AIoT)

- \* A minor in 5G & AIoT would be awarded if a student completes 4 elective modules including the mandatory module (EP0409), at least 1 module from AI domain (EP0609, EP0303), at least 1 module from IoT domain (EP0401, EP0403), and/or EP0411.