Department of Electronic Engineering
City University of Hong Kong
EE5415 Mobile Applications Design and Development

Lab03: Layouts and Widgets (Java)

Students are required to demonstrate their implemented apps to obtain the marks. When you have completed one of these steps, simply raise your hand and demonstrate the step to the instructor who will then initial the **Instructor Verification** line for that step.

- 5 marks for Linear Layout App
- 5 marks for Constraint Layout App
- 5 marks for Table Layout App
- 20 marks for "Form Stuff App" with 6 widgets (**Image button, Edit Text, Check Box, Radio Button, Toggle Button, Rating Bar**)
- 15 marks for enhancement of the BMI app to support landscape display
- 15 marks for implementing "About BMI" button in BMI app
- 35 marks for short questions

I. Introduction

The Android platform supplies a rich set of components for building graphical user interfaces (GUIs) on mobile devices. The user interface (UI) widgets are called views in Android terminology. Layout is the architecture for the UI in an Android Activity. It defines the layout structure and holds all the elements that appear to the user. You can declare your layout in two ways:

1. **Declare UI elements in XML.** Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

2. **Instantiate layout elements at runtime.** Your application can also create `View` and `ViewGroup` objects (and manipulate their properties) programmatically.

The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior. Your UI descriptions are external to your application code, which means that **you can modify or adapt it without having to modify your source code and recompile**. For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems. As such, there are some exercises in this lab focus on teaching you how to declare common layouts in XML.

In addition, the Android platform provides a rich mix of widgets that optimized for mobile devices. In Android, a widget is a `View` object that serves as an interface for interaction with the user. Thus, these widgets are called views in Android and represented by subclasses of the `android.view.View` class. Nearly all of them belong to the `android.widget` package.

Android provides a set of fully implemented widgets, like buttons, checkboxes, and text-entry fields (edit texts), so you can quickly build your UI. Some widgets provided by Android are more complex, like a date picker, a clock, and zoom controls. But you are not limited to the kinds of widgets provided by the Android platform. If you would like to do something more customized and create your own actionable elements, you can, by defining your own View object or by extending and combining existing widgets.

II. Layouts

2.1 Linear Layout

`LinearLayout` is a `ViewGroup` that displays child `View` elements in a linear direction, either **vertically** or **horizontally**. You should be careful about over-using the `LinearLayout`. If you begin nesting multiple Linear Layouts, you may want to consider using a `ConstraintLayout` instead (see next section).

1. Start a new project named **Linear Layout**.
2. Follow the hello-world-style to fill in these values:
   - Choose your project : **Empty Activity**
   - Application name : **Linear Layout**
   - Package name : **com.example.linearlayout**
   - Language : Java
   - Minimum API level: API 19: Android 4.4 (KitKat)
   - Click Finish
   - Default created Activity:
     - Activity Name: **MainActivity**
     - Layout Name: **activity_main**
3. Modify the layout XML in **`res/layout/activity_main.xml`** with following content using LinearLayout:

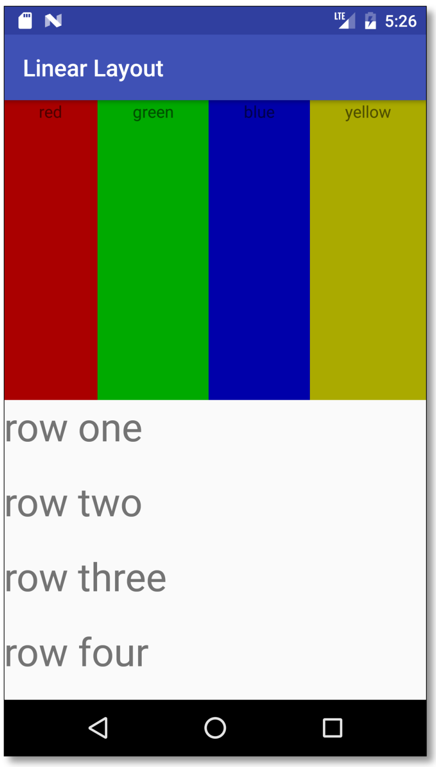
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent >

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1">
        <TextView
            android:text="red"
            android:gravity="center_horizontal"
            android:background="#ffaa0000"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
        <TextView
            android:text="green"
            android:gravity="center_horizontal"
            android:background="#ff00aa00"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
        <TextView
            android:text="blue"
            android:gravity="center_horizontal"
            android:background="#0000aa"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
        <TextView
            android:text="yellow"
            android:gravity="center_horizontal"
            android:background="#ffaaaa00"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
    </LinearLayout>
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1">
        <TextView
            android:text="row one"
            android:textSize="15pt"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
        <TextView
            android:text="row two"
            android:textSize="15pt"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
        <TextView
            android:text="row three"
            android:textSize="15pt"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
        <TextView
            android:text="row four"
            android:textSize="15pt"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
    </LinearLayout>
</LinearLayout>
```

Carefully inspect this XML. There is a root `LinearLayout` that defines its **orientation** to be **vertical**—all child Views (of which it has two) will be stacked vertically. The first child is another `LinearLayout` that uses a **horizontal orientation** and the second child is a `LinearLayout` that uses a vertical orientation. Each of these nested `LinearLayouts` contains several `TextView` elements, which are oriented with each other in the manner defined by their parent `LinearLayout`.

4. By default the `onCreate()` method in the `MainActivity.java` is pointing to the XML layout file located at `res/layout/activity_main.xml`. The `setContentView(int)` method loads the layout file for the Activity, specified by the resource ID — `R.layout.activity_main` refers to the `res/layout/activity_main.xml` layout file.

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

5. Run the application. You should see the following:



row one

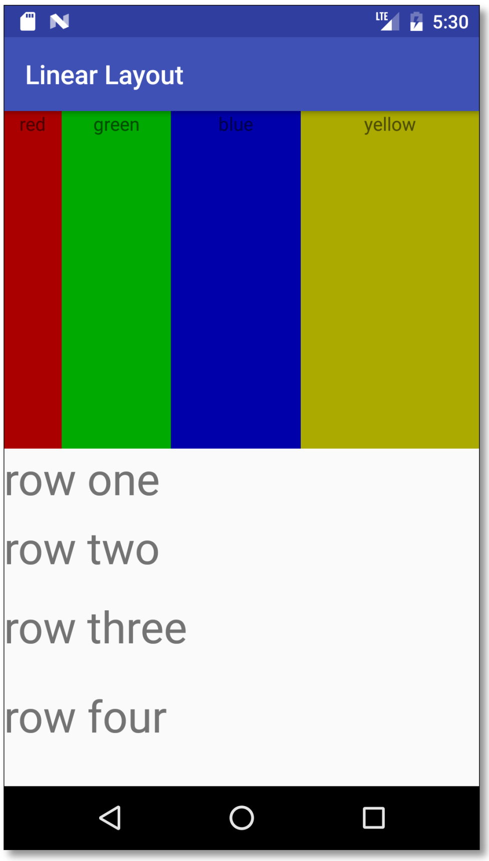
row two

row three

row four

Notice how the XML attributes define each View's behavior. Try experimenting with different values for **`android:layout_weight`** to see how the screen real estate is distributed based on the weight of each element.

6. Pay particular attention to the `layout_weight` attribute. Then create this layout:



The legal values for this attribute are documented here:

http://developer.android.com/guide/topics/resources/layout-resource.html

2.2 Constraint Layout

`ConstraintLayout` is a new layout of the `ViewGroup`. `ConstraintLayout` is much more flexible than `RelativeLayout` or `LinearLayout`. Basically, it is a flexible layout manager for your app that allows you to create dynamic user interfaces without nesting multiple layouts. It is distributed as an **androidx** library that is tightly coupled with Android Studio and backwards compatible to API Level 9.

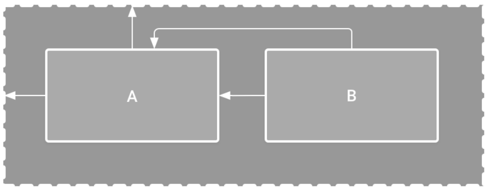
1. Create a new project with application name of "**Constraint Layout"** and modify the `activity_layout.xml` inside the `res/layout/` folder with following content:

```
<?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=".MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView" />
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="36dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="24dp"
        android:text="Type here:"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:layout_marginTop="16dp"
        android:text="OK"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:layout_marginTop="16dp"
        android:text="CANCEL"
        app:layout_constraintEnd_toStartOf="@+id/button"
        app:layout_constraintTop_toBottomOf="@+id/editText" />

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

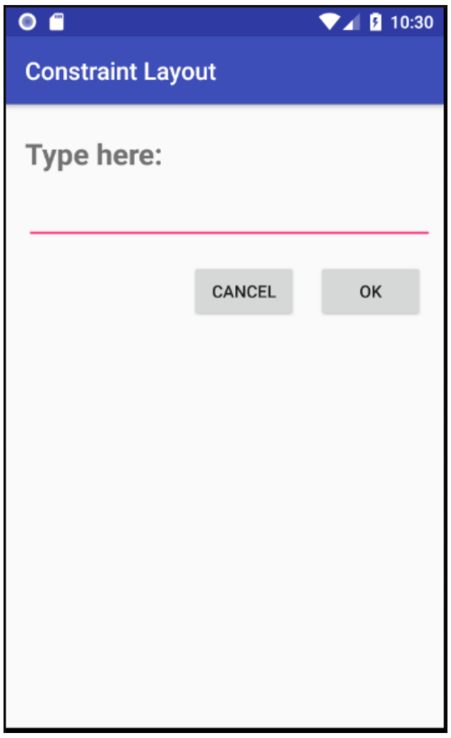
Notice each of the `android:layout_*` attributes, such as `layout_marginEnd`, `layout_marginStart`, `layout_marginTop`. and `layout_marginBottom`. When using a `ConstraintLayout`, you can use these attributes to describe how you want to position each `View`. `ConstraintLayout` is a layout that defines the position for each view based on constraints to sibling views and the parent layout. In this way, you can create both simple and complex layouts with a flat view hierarchy. That is, it avoids the need for nested layouts, which can increase the time required to draw the UI.

For example, you can declare the following layout:

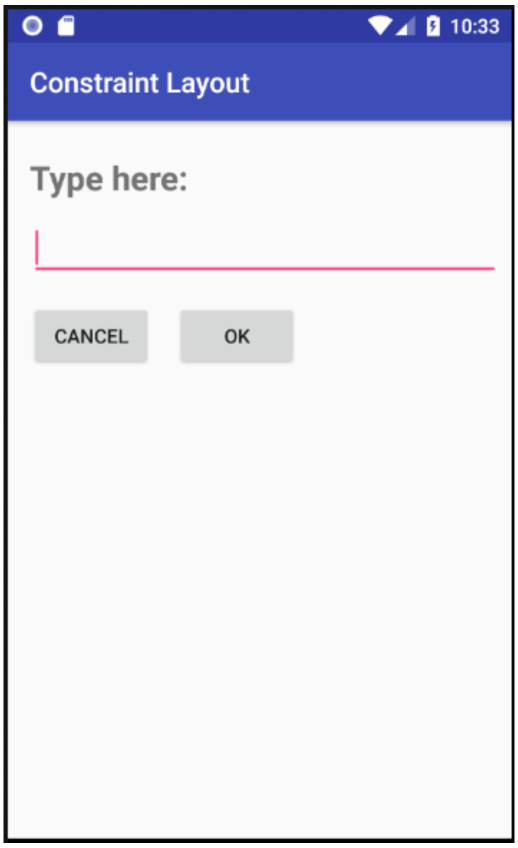
- View A appears 16dp from the top of the parent layout.
- View A appears 16dp from the left of the parent layout.
- View B appears 16dp to the right of view A.
- View B is aligned to the top of view A.



2. Run the applications. You should see the following:



3. Modify the code of `activity_main.xml` to make the UI with these two button on the left side as shown below:



2.3 Table Layout

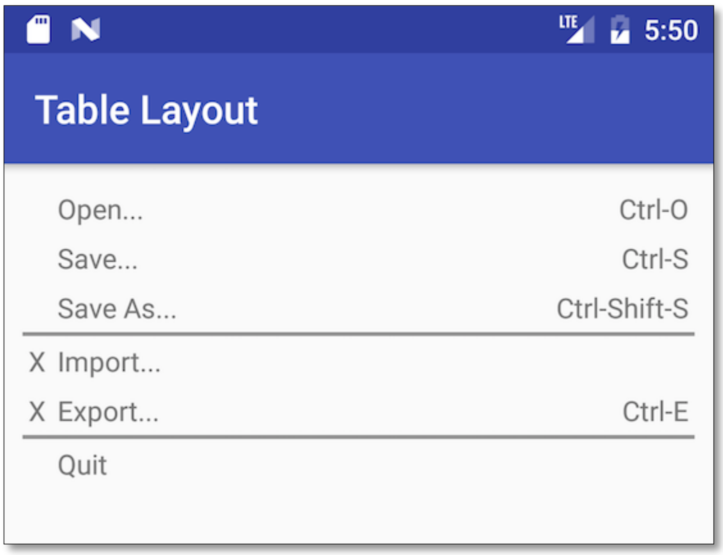
TableLayout is a ViewGroup that displays child View elements in rows and columns. A TableLayout consists of a number of TableRow objects, each defining a row (actually, you can have other children, which will be explained below). TableLayout containers do not display border lines for their rows, columns, 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.

1. To learn how to use TableLayout, create a new project with application name of "**Table Layout**" and modify the activity_layout.xml inside the res/layout/ folder with following content:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:stretchColumns="1">

    <TableRow>
        <TextView
            android:layout_column="1"
            android:padding="3dp"
            android:text="Open..." />
        <TextView
            android:gravity="right"
            android:padding="3dp"
            android:text="Ctrl-O" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:padding="3dp"
            android:text="Save..." />
        <TextView
            android:gravity="right"
            android:padding="3dp"
            android:text="Ctrl-S" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout_column="1"
            android:padding="3dp"
            android:text="Save As..." />
        <TextView
            android:gravity="right"
            android:padding="3dp"
            android:text="Ctrl-Shift-S" />
    </TableRow>

    <View
        android:layout_height="2dp"
        android:background="#FF909090" />

    <TableRow>
        <TextView
            android:padding="3dp"
            android:text="X" />
        <TextView
            android:padding="3dp"
            android:text="Import..." />
    </TableRow>

    <TableRow>
        <TextView
            android:padding="3dp"
            android:text="X" />
        <TextView
            android:padding="3dp"
            android:text="Export..." />
        <TextView
            android:gravity="right"
            android:padding="3dp"
            android:text="Ctrl-E" />
    </TableRow>

    <View
        android:layout_height="2dp"
        android:background="#FF909090" />

    <TableRow>
        <TextView
            android:layout_column="1"
            android:padding="3dp"
            android:text="Quit" />
    </TableRow>
</TableLayout>
```

Notice how this resembles the structure of an HTML table. The `TableLayout` element is like the HTML `<table>` element; `TableRow` is like a `<tr>` element; but for the cells, you can use any kind of `View` element. In this example, a `TextView` is used for each cell. In between some of the rows, there is also a basic `View`, which is used to draw a horizontal line.

2. Run the applications. You should see the following:



```
5:50
Table Layout

    Open...                  Ctrl-O
    Save...                  Ctrl-S
    Save As...         Ctrl-Shift-S
-----------------------------------
X   Import...
X   Export...                Ctrl-E
-----------------------------------
    Quit
```

Check Point 1:

1.1 Show your Linear Layout Project with use of different weights. 1.2 Show your Constraint Layout Project with buttons aligns to the left. 1.3 Show your Table Layout Project.
Instructor Verification (separate page)

III. Android Widget

In Android, a widget is a `View` object that serves as an interface for interaction with the user. Android provides a set of fully implemented widgets, like buttons, checkboxes, and text-entry fields, so you can quickly build your UI. Some widgets provided by Android are more complex, like a date picker, a clock, and zoom controls. But you are not limited to the kinds of widgets provided by the Android platform. If you would like to do something more customized and create your own actionable elements, you can, by defining your own `View` object or by extending and combining existing widgets.

3.1 Form Stuff App

This part introduces a variety of widgets that are useful when creating forms, such as image buttons, text fields, checkboxes and radio buttons.

1. Start a new project with application name of **“Form Stuff”**.
2. Your `res/layout/activity_main.xml` file should already have a basic `ConstraintLayout`. Replace this `ConstraintLayout` by a `LinearLayout` as follows:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

</LinearLayout>
```

For each widget you are going to add, just put the respective View inside this Linear Layout.

Each section below also assumes that your `MainActivity` Activity has the following default implementation of the `onCreate()` method:

```
public class MainActivity extends AppCompatActivity {

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

     }
}
```

3.2 Image Button

In this section, you will create an Image Button with a custom image instead of text, using the Button widget and an XML file that defines three different images to use for the different button states. When the button is pressed, a short message will be displayed.

1. Copy the three images into the **`res/drawable/`** directory of your project. These will be used for the different button states.
   - `android_focused.png`
   - `android_normal.png`
   - `android_pressed.png`

(*These images can be found in the images.zip that can be downloaded from the course schedule webpage.)*

2. Create a new Drawable resource file named **android_button.xml** within the **`res/drawable`**/ folder by *right click the drawable folder =>New=>Drawable resource file*. Then insert the following XML code in this file (`res/drawable/android_button.xml`):

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

    <item android:drawable="@drawable/android_pressed"
        android:state_pressed="true" />
    <item android:drawable="@drawable/android_focused"
        android:state_focused="true" />
    <item android:drawable="@drawable/android_normal" />

</selector>
```

This defines a single drawable resource, which will change its image based on the current state of the button. The first `<item>` defines `android_pressed.png` as the image when the button is pressed (it's been activated); the second `<item>` defines `android_focused.png` as the image when the button is focused (when the button is highlighted using the trackball or directional pad); and the third `<item>` defines android_normal.png as the image for the normal state (when neither pressed nor focused). This XML file now represents a single drawable resource and when referenced by a Button for its background, the image displayed will change based on these three states.

3. Open the `res/layout/activity_main.xml` file and add the Button element:

```
<ImageButton
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:src="@drawable/android_button" />
```

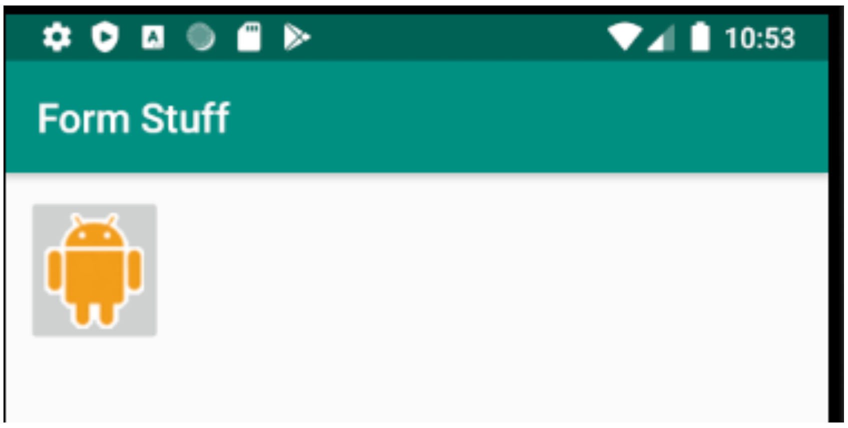
The **android:src** attribute specifies the drawable resource to use for the button background (which, when saved at `res/drawable/android_button.xml`, is referenced as `@drawable/android_button`). This replaces the normal background image used for buttons throughout the system. In order for the drawable to change its image based on the button state, the image must be applied to the background.

4. To make the button do something when pressed, add the following code at the end of the `onCreate()` method:

```
final ImageButton button = (ImageButton)findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Perform action on clicks
        Toast.makeText(MainActivity.this, "Beep Bop", Toast.LENGTH_SHORT).show();
    }
});
```

This function is used to display a toast message when the button is pressed.

5. Now run the application and press the image button to experience the effect of changing the image of the button.



Press the image button and observer change of the button and the Toast message with text of "Beep Bop".

3.3 Edit Text

1. Open the **res/layout/activity_main.xml** file and add the EditText element (inside the LinearLayout):

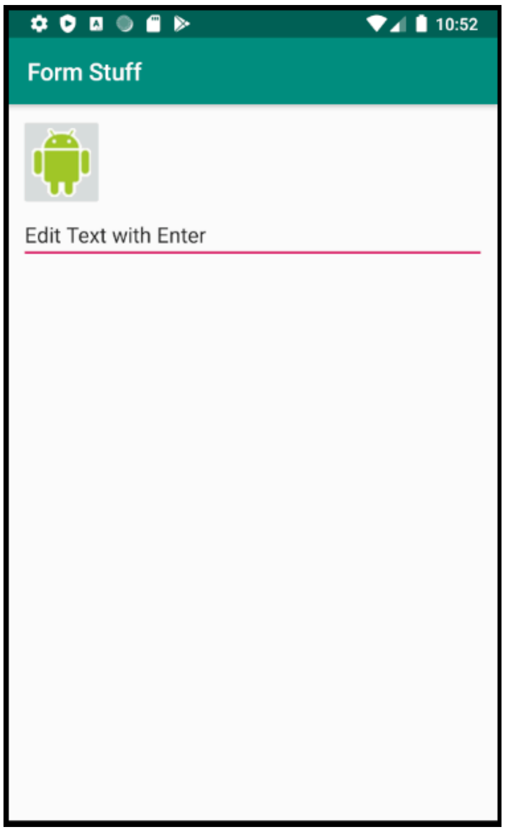
```
<EditText
    android:id="@+id/editText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

2. To do something with the text that the user types, add the following code to the end of the `onCreate()` method:

```
final EditText edittext = (EditText) findViewById(R.id.editText);

edittext.setOnKeyListener(new View.OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        // If the event is a key-down event on the "enter" button
        if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
                (keyCode == KeyEvent.KEYCODE_ENTER)) {
            // Perform action on key press
            Toast.makeText(MainActivity.this, edittext.getText(),
Toast.LENGTH_SHORT).show();
            return true;
        }
        return false;
    }
});
```

This captures the `EditText` element from the layout and adds an `View.OnKeyListener`. The `View.OnKeyListener` must implement the `onKey(View, int, KeyEvent)` method, which defines the action to be made when a key is pressed while the widget has focus. In this case, the method is defined to listen for the Enter key (when pressed down), then pop up a Toast message with the text that has been entered. The `onKey(View, int, KeyEvent)` method should always return true if the event has been handled, so that the event doesn't bubble-up (which would result in a carriage return in the text field).

3. Run the application. Ending some texts on the text field and then **press the "Enter" Key** to show the Toast message as shown below:



3.4 Check Box

In this section, you will create a checkbox for selecting items, using the `CheckBox` widget. When the checkbox is pressed, a toast message will indicate the current state of the checkbox.

1. Open the **`res/layout/activity_main.xml`** file and add the `CheckBox` element (inside the `LinearLayout`):

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:text="Single checkbox example" />

<CheckBox
    android:id="@+id/cb_single"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CheckBox" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Select your favorite shoes brands" />

<CheckBox
    android:id="@+id/cb_addidas"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onCheckboxClicked"
    android:text="Adidas" />

<CheckBox
    android:id="@+id/cb_nike"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onCheckboxClicked"
    android:text="Nike" />
```

2. To add the ckeckbox OnClickListener for single checked box **inside** the `onCreate()` method of the MainActivity.java:

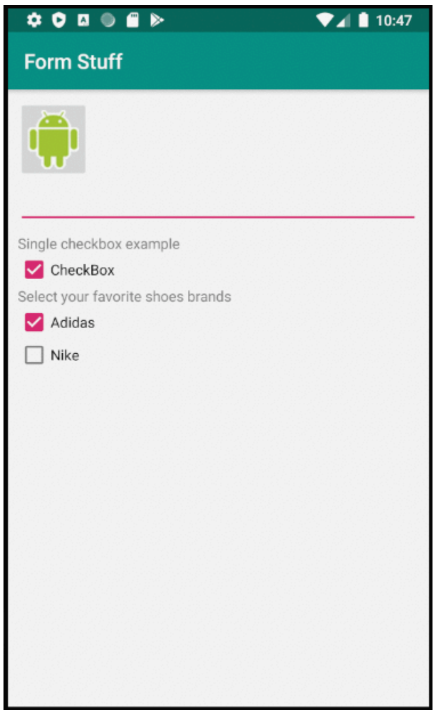
```
final CheckBox cb_single = (CheckBox) findViewById(R.id.cb_single);

cb_single.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Perform action on clicks
        if (cb_single.isChecked()) {
            Toast.makeText(MainActivity.this, "Ckecker", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(MainActivity.this, "UnCkecker", Toast.LENGTH_LONG).show();
        }
    }
});
```

3. To do something when the state of the multiple checkboxes is changed, add the following code to in the MainActivity.java but **outside** the `onCreate()` method:

```
// multiple checkbox click method
public void onCheckboxClicked(View view) {
    // Is the button now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.cb_addidas:
            if (checked)
                Toast.makeText(MainActivity.this, "Addida Selected",
                        Toast.LENGTH_SHORT).show();
            else
                Toast.makeText(MainActivity.this, "Addida Unselected",
                        Toast.LENGTH_SHORT).show();
            break;
        case R.id.cb_nike:
            if (checked)
                Toast.makeText(MainActivity.this, "Nike Selected",
                        Toast.LENGTH_SHORT).show();
            else
                Toast.makeText(MainActivity.this, "Nike Unselected",
                        Toast.LENGTH_SHORT).show();
            break;
    }
```

The method you declare in the `android:onClick` attribute must have a signature exactly as shown above. This captures the `CheckBox` element from the layout, then adds an `onCheckboxClicked()` method in the `MainActivity.java` Class. When the checkbox is clicked, the `onCheckboxClcked()` is called to check the new state of the check box. If it has been checked, then a Toast displays the message "Selected", otherwise it displays "Removed".

4. Run it.


10:47
Form Stuff
Single checkbox example
☑ CheckBox
Select your favorite shoes brands
☑ Adidas
☐ Nike

3.5 Radio Button

In this section, you will create two mutually-exclusive radio buttons (enabling one disables the other), using the `RadioGroup` and `RadioButton` widgets. When either radio button is pressed, a toast message will be displayed.

1. Open the **`res/layout/activity_main.xml`** file and add two `RadioButtons`, nested in a `RadioGroup` (inside the `LinearLayout`):

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

    <RadioButton
        android:id="@+id/radio_red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Red"
        android:onClick="onRadioButtonClicked"/>

    <RadioButton
        android:id="@+id/radio_blue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Blue"
        android:onClick="onRadioButtonClicked"                                  />
</RadioGroup>
```

It's important that the `RadioButtons` are grouped together by the `RadioGroup` element so that no more than one can be selected at a time. This logic is automatically handled by the Android system. When one `RadioButton` within a group is selected, all others are automatically deselected.

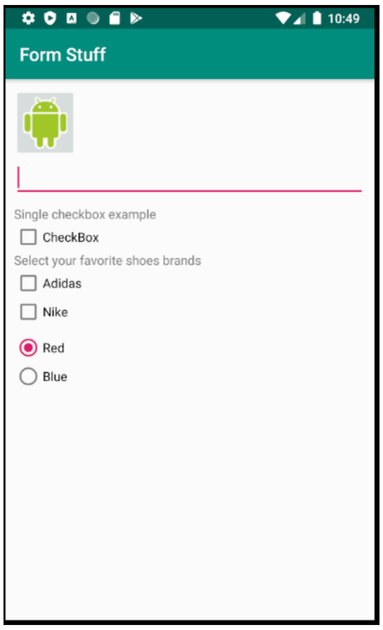
2. To do something when each `RadioButton` is selected, we use the the `android:onClick` attribute in the RadioButton tag with “`onRadioButtonClicked`” field. In this case, we need to create a `onRadioButtonClicked()` method in the MainActivity.java Class to response when these radio buttons are selected. Such that we have to add the following code in the MainActivity.java outside the `onCreate()` method.

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_red:
            if (checked)
                Toast.makeText(MainActivity.this, "Red Selected",
                        Toast.LENGTH_SHORT).show();
            break;
        case R.id.radio_blue:
            if (checked)
                Toast.makeText(MainActivity.this, "Blue Selected",
                        Toast.LENGTH_SHORT).show();
            break;
    }
}
```

First, the View that is passed to the `onRaidoButtonClick(View)` method is cast into a

RadioButton. Then a Toast message displays the selected radio button's text.

3. Run the application.



3.6 Toggle Button

In this section, you will create a button used specifically for toggling between two states, using the ToggleButton widget. This widget is an excellent alternative to radio buttons if you have two simple states that are mutually exclusive ("on" and "off", for example).

1. Open the res/layout/activity_main.xml file and add the ToggleButton element:

```
<ToggleButton android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Vibrate on"
    android:textOff="Vibrate off"/>
```

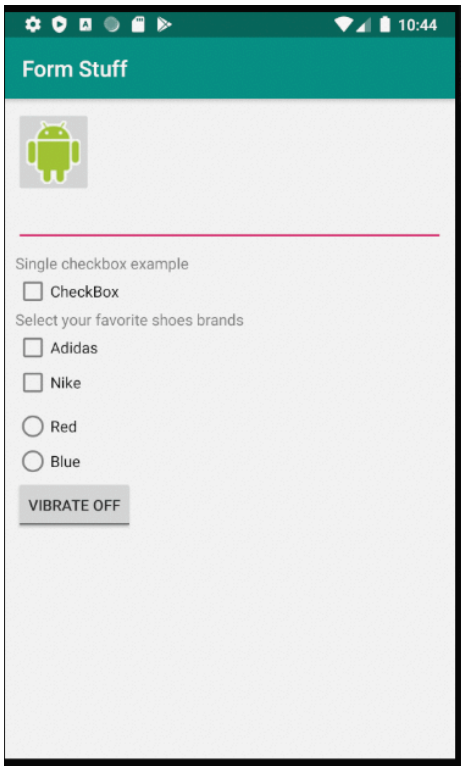
2. To do something when the state is changed, add the following code to the end of the onCreate() method:

```
final ToggleButton togglebutton = (ToggleButton) findViewById(R.id.toggleButton);
togglebutton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        if (togglebutton.isChecked()) {
            Toast.makeText(MainActivity.this, "Toggle Button Checked",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(MainActivity.this, "Toggle Button Not checked",
Toast.LENGTH_SHORT).show();
        }
    }
});
```

This captures the ToggleButton element from the layout, then adds an

`View.OnClickListener`. The `View.OnClickListener` must implement the `onClick(View)` callback method, which defines the action to perform when the button is clicked. In this example, the callback method checks the new state of the button, then shows a Toast message that indicates the current state. Notice that the ToggleButton handles its own state change between checked and unchecked, so you just ask which it is.

3. Run the application.



3.7 Rating Bar

Students will create a `RatingBar` widget that allows the user to provide a rating.

1. Open the `res/layout/activity_main.xml` file and add the `RatingBar` element:

```
<RatingBar android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:stepSize="1.0"/>
```

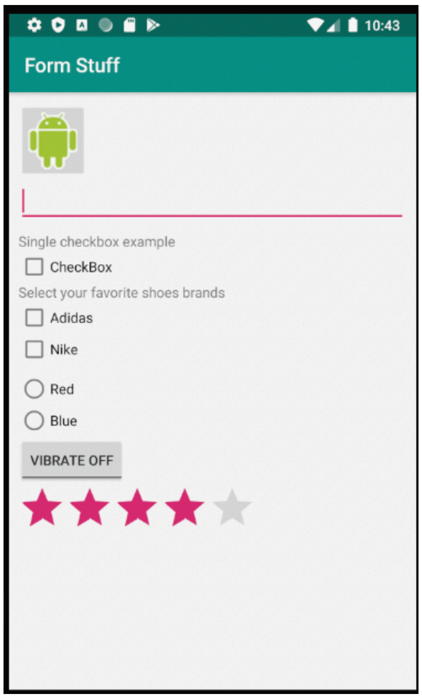
The `android:numStars` attribute defines how many stars to display for the rating bar. The `android:stepSize` attribute defines the granularity for each star (for example, a value of 0.5 would allow half-star ratings).

2. To do something when a new rating has been set, add the following code to the end of the `onCreate()` method:

```
final RatingBar ratingbar = (RatingBar) findViewById(R.id.ratingBar);

ratingbar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean
fromUser) {
        Toast.makeText(MainActivity.this, "New Rating: " + rating,
Toast.LENGTH_SHORT).show();
    }
});
```

This captures the `RatingBar` widget from the layout with `findViewById(int)` and then sets an `RatingBar.OnRatingBarChangeListener`. The `onRatingChanged()` callback method then defines the action to perform when the user sets a rating. In this case, a simple Toast message displays the new rating.

3. Run the application. If you've added all the form widgets above, your application should look like this:



Check Point 2:

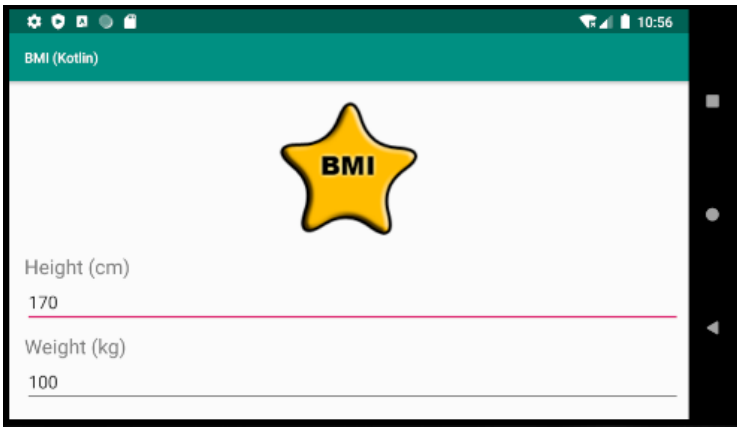
2. Run your "Form Stuff App" with following 6 widgets:
- **Image button**
- **Edit Text**
- **Check Box**
- **Radio Button**
- **Toggle Button**
- **Rating Bar**

Instructor Verification (separate page)

IV. Enhancements of BMI App

4.1 Supporting Landscape Display Mode

In this section, students are required to re-build the BMI project with basic features of the Lab02 and make it supports Landscape display mode using "Constraint Layout". Unlike desktop applications, mobile apps usually have two UIs for portrait and landscape orientation. You can press **Ctrl-F12** to rotate the screen of your emulator. You will find that the BMI layouts defined in Lab02 can fit the landscape screen display mode with the "SEE BMI REPORT NOW" button outside the screen as shown below.



In Android, you can define multiple sets of resources that the system will automatically adapt to when configuration is changed.

1. Make two copies of the res/layout folder. Rename the folders as below:
    - **res/layout-land** (alternative resource for Landscape Mode)
    - **res/layout-port** (alternative resource for Portrait Mode)

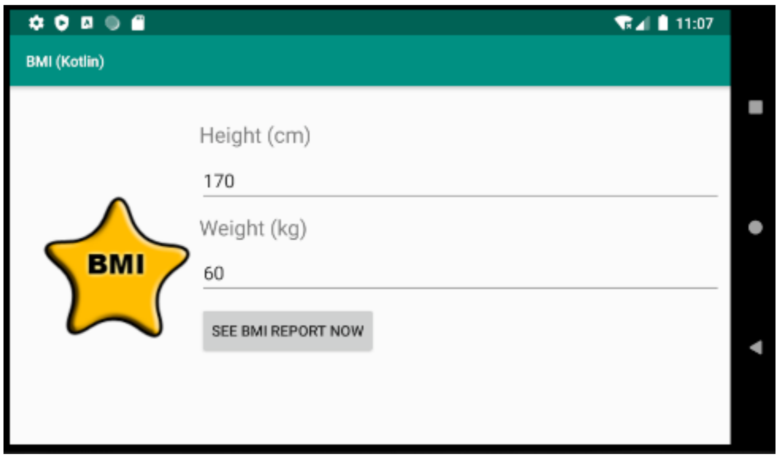
The original res/layout folder remains as the default resource when no alternatives can be matched.

2. First create a new folder of `res/layout-land/`, within this new folder create a new file of `activity_main.xml` and insert the following layout XML code:

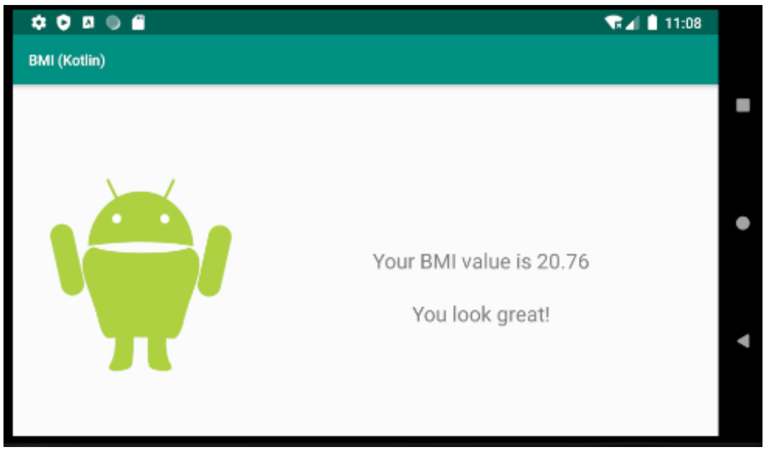
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
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

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="140dp"
        android:layout_height="140dp"
        android:layout_marginStart="32dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/icon_128" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="32dp"
        android:text="@string/bmi_height"
        android:textSize="20sp"
        app:layout_constraintStart_toEndOf="@+id/imageView"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/heightET"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="number"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/imageView"
        app:layout_constraintTop_toBottomOf="@+id/textView" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="@string/bmi_weight"
        android:textSize="20sp"
        app:layout_constraintStart_toEndOf="@+id/imageView"
        app:layout_constraintTop_toBottomOf="@+id/heightET" />

    <EditText
        android:id="@+id/weightET"
        android:layout_width="0dp"
        android:layout_height="44dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="number"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/imageView"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />

    <Button
        android:id="@+id/reportBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="@string/bmi_btn"
        app:layout_constraintStart_toEndOf="@+id/imageView"
        app:layout_constraintTop_toBottomOf="@+id/weightET" />

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

3. Similarly, create a new `res/layout-land/activity_report.xml` file and re-arrange the UI elements in the interface to support landscape display mode.

4. Run the application again. You should see something like this:



Height (cm)
170
Weight (kg)
60



Your BMI value is 20.76

You look great!

Check Point 3:

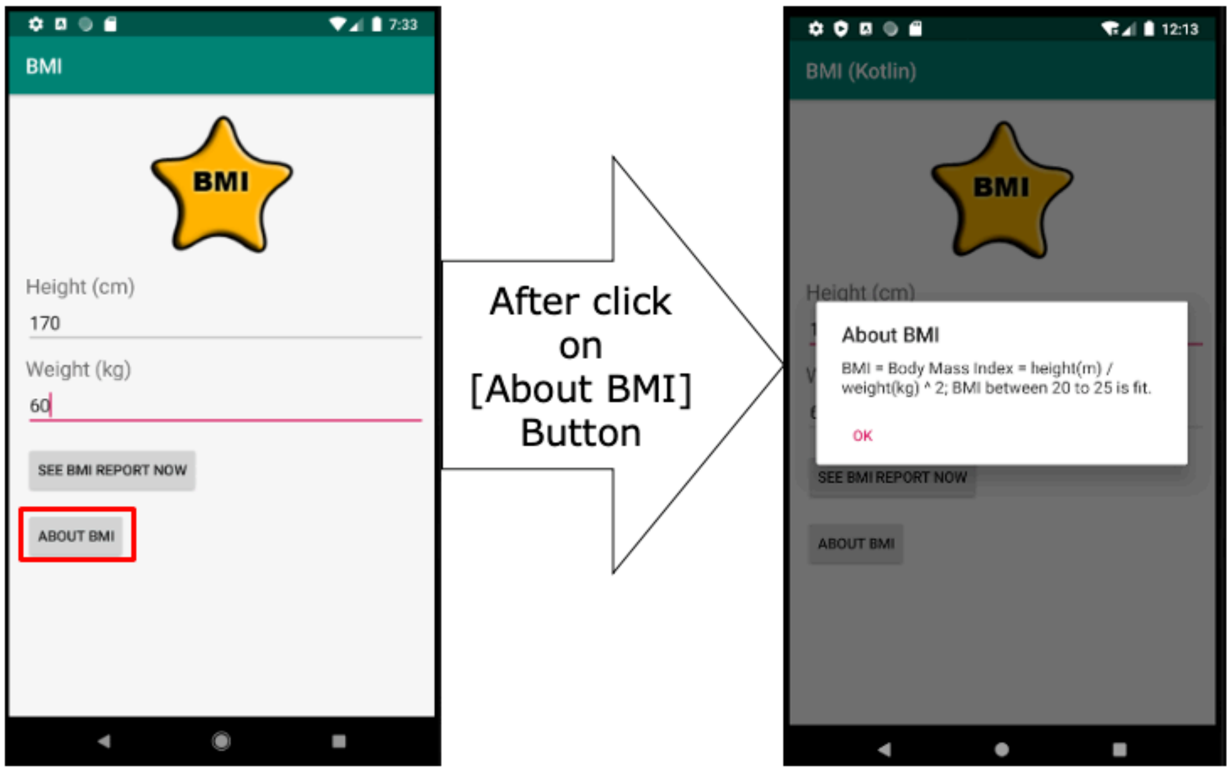
3. Running your BMI app that support landscape mode.
Instructor Verification (separate page)

5. This step is Optional for student to experience Linear Layout approach to solve the Landscape mode problem. Instead of using Constraint Layout, we can also use the Linear Layout to create a similar Landscape user interface. Replace the `res/layout-land/activity_main.xml` with follow code, which use two Linear Layouts with horizontal and vertical orientations to achieve similar UI components arrangement as the Contraint Layout approach.

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

    <ImageView android:id="@+id/imageView"
        android:layout_width="140dp"
        android:layout_height="140dp"
        android:layout_gravity="center_vertical"
        android:paddingTop="10dp"
        android:paddingBottom="10dp"
        android:src="@drawable/icon_128"
        android:layout_weight="1"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:gravity="center_vertical"
        android:layout_weight="3">
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/bmi_height"
            android:textSize="20sp" />

        <EditText android:id="@+id/heightET"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text=""
            android:layout_marginBottom="10dp" />

        <TextView android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/bmi_weight"
            android:textSize="20sp" />

        <EditText android:id="@+id/weightET"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="10dp" />

        <Button android:id="@+id/reportBtn"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/bmi_btn"
            android:onClick="calcBMI"    />

    </LinearLayout>

</LinearLayout>
```

4.2 About Button using Dialog

In this section, students are required to further enhance the BMI app to provide a help function by an "About BMI" button and a dialog for showing the information about the BMI as shown below:



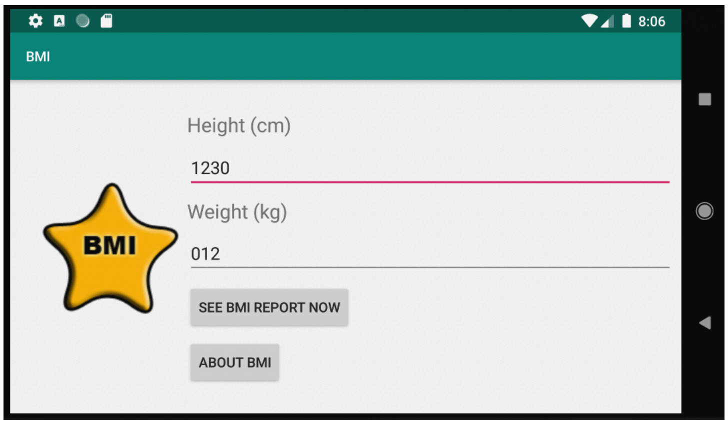
BMI = Body Mass Index = height(m) / weight(kg) ^ 2; BMI between 20 to 25 is fit.

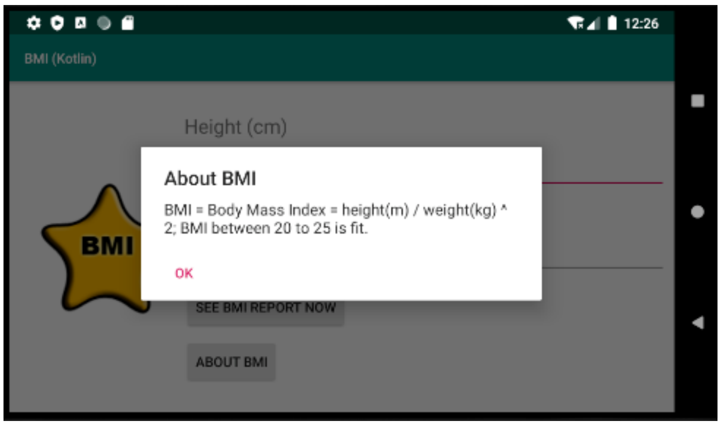
The following codes can add a button the show this dialog.

In `res/layout-port/activity_main.xml`, use the following code to add the "About BMI" button for portrait display mode but you also need to add the position constraints to make the button located in the correct position :

```
<Button
    android:id="@+id/aboutBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:text="@string/about_button"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/reportBtn" />
```

However, you have to find out how to add the "About BMI" button in the **`res/layout-land/activity_main.xml`** using `ConstraintLayout` for Landscape Mode under the folder of layout-land as show below:





About BMI

BMI = Body Mass Index = height(m) / weight(kg) ^ 2; BMI between 20 to 25 is fit.

OK

In `res/values/strings.xml`, use the following code to add the information of the "About BMI".

```
<string name="about_button">About BMI</string>
<string name="about_msg">BMI = Body Mass Index = height(m) / weight(kg) ^
2; BMI between 20 to 25 is fit.</string>
```

Modify the `MainActivity.java` for adding the "About BMI" button method of **`aboutButton.setOnClickListener()`**:

```
aboutButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                AlertDialog.Builder builder =
                       new AlertDialog.Builder(MainActivity.this);
                builder.setTitle(R.string.about_button);
                builder.setMessage(R.string.about_msg);
                builder.setPositiveButton("OK",
                       new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which)
                        {  }
                });
                builder.create();
                builder.show();
            }
        });
```

In order to the AlertDialog, students should make sure that `android.app.AlertDialog` is imported in the `MainActivity.java`.

```
import android.app.AlertDialog;
```

Check Point 4:

Running your BMI app with an “About BMI” button for both portrait and landscape modes.
Instructor Verification (separate page)