Department of Electronic Engineering

City University of Hong Kong

EE5415 Mobile Applications Design and Development

Lab04: Localization and Menus

I. Supporting Different Devices

Android devices come in many shapes and sizes all around the world. With a wide range of devices, you have an opportunity to reach a huge audience with your app. In order to be as successful as possible on Android, your app needs to adapt to various device configurations. Some of the important variations that you should consider include different languages, screen sizes, and versions of the Android platform.

This lab teaches you how to use basic platform features that leverage alternative resources and other features so your app can provide an optimized user experience on a variety of Android-compatible devices, using a single application package (APK).

1.1 Supporting Different Languages

It's always a good practice to extract UI strings from your app code and keep them in an external file. Android makes this easy with a resources directory in each Android project. If you created your project using the Android SDK Tools (read Creating an Android Project), the tools create a **`res/`** directory in the top level of the project. Within this **`res/`** directory are subdirectories for various resource types. There are also a few default files such as **`res/values/strings.xml`**, which holds your string values.

1.1.1 Create Locale Directories and String Files

To add support for more languages, create additional values directories inside **`res/`** that include a hyphen and the ISO country code at the end of the directory name. For example, **`values-zh`**/ is the directory containing simple resources for the Locales with the language code "**zh**". Android loads the appropriate resources according to the locale settings of the device at run time.

Once you've decided on the languages you will support, create the resource subdirectories and string resource files. For example:

English (default locale), **res/values/strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name" >My Application</string>
    <string name="hello_world" >Hello World!</string>
</resources>
```

Simplified Chinese character, **res/values-zh-rCN/strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name" >我的应用程序</string>
    <string name="hello_world" >你好世界！（中国）</string>
</resources>
```

Traditional Chinese character, **res/values-zh-rHK/strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name" >我的應用程序</string>
    <string name="hello_world" >你好世界！（中國香港）</string>
</resources>
```

French, **res/values-fr/strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name" >Mon Application</string>
    <string name="hello_world" >Bonjour le monde !</string>
</resources>
```

Note: You can use the locale qualifier (or any configuration qualifier) on any resource type, such as if you want to provide localized versions of your bitmap drawable. For more information, see

- Localization: https://developer.android.com/guide/topics/resources/localization.html

1.1.2 Use the String Resources

You can reference your string resources in your source code and other XML files using the resource name defined by the **`<string>`** element's name attribute. In your source code, you can refer to a string resource with the syntax **`R.string.<string_name>`**. There are a variety of methods that accept a string resource this way. For example:

```
// Get a string resource from your app's Resources
        String hello = getResources().getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
        TextView textView = new TextView(this);
        textView.setText(R.string.hello_world);
```

In other XML files, you can refer to a string resource with the syntax **`@string/<string_name>`** whenever the XML attribute accepts a string value. For example:

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

1.1.2 Chinese Localization

In Android Studio, to start a new project as follows:

- In Android Studio, create a new project called "Chinese Localization" with the following values:
  - **Empty Activity** Project
  - Application name : **Chinese Localization**
  - Package name : **com.example.chineselocalization**
  - Minimum API level: **API 19: Android 4.4 (KitKat)**
  - Language: **Java**
  - Activity name: **MainActivity**
  - Layout name: **activity_main**

- Create the folders of **`/res/values-zh-rCN`** and **`/res/values-zh-rHK`** with corresponding `strings.xml` files to make it support Simplified Chinese and Traditional Chinese.

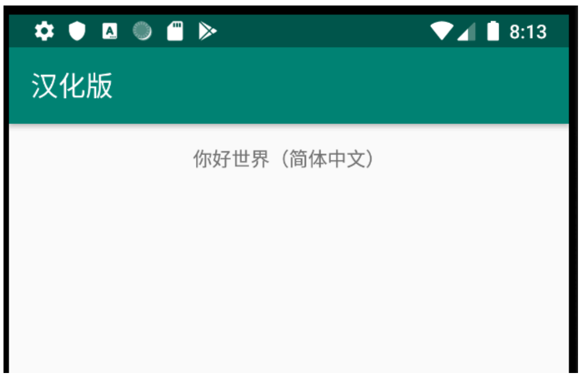
Simplified Chinese character, **`res/values-zh-rCN/strings.xml`**:

```
<resources>
    <string name="app_name">汉化版</string>
    <string name="hello_world">你好世界（简体中文）</string>
</resources>
```

Traditional Chinese character, **`res/values-zh-rHK/strings.xml`**:

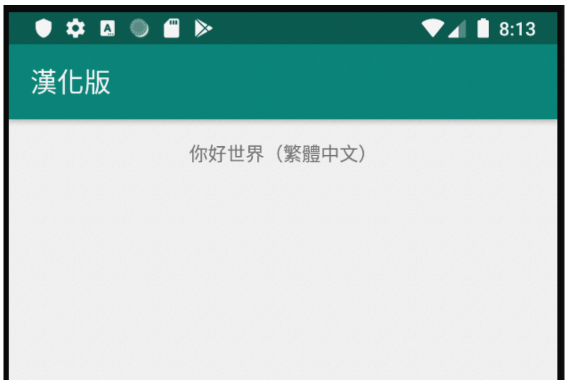
```
<resources>
    <string name="app_name">漢化版</string>
    <string name="hello_world">你好世界（繁體中文）</string>
</resources>
```

- Run the project and change the language of the emulator (**Setting -> Language & input -> Language**) to these languages 中文（简体）and 中文（繁體）with results as shown below:


8:13

汉化版

你好世界（简体中文）

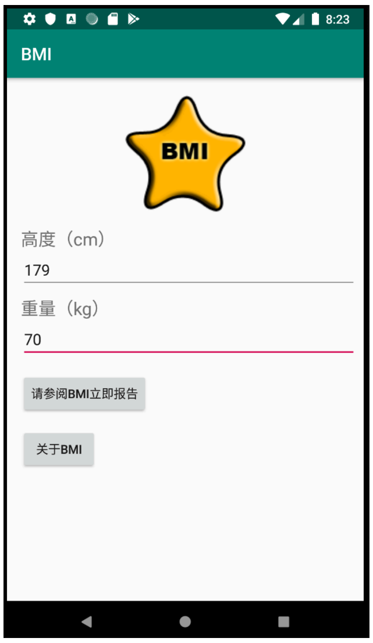

8:13

漢化版

你好世界（繁體中文）

1.1.3 Chinese Localization of BMI Project

Re-build the BMI project with basic features of the Lab03 and make it supports "Simplified Chinese" and the region of Mainland China using a folder of "`res/values-zh-rCN/`". Run your completed project and demonstrate the UI as shown below:

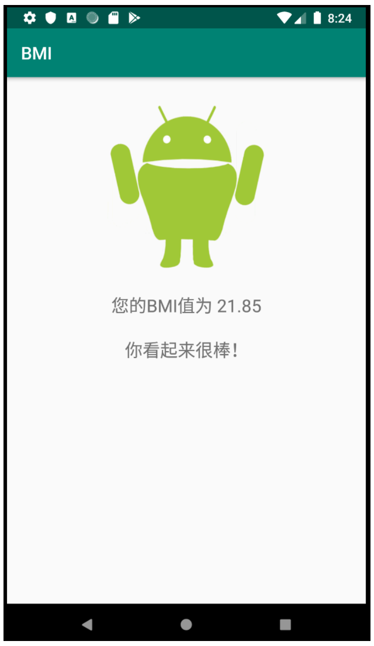

BMI

高度（cm）

179

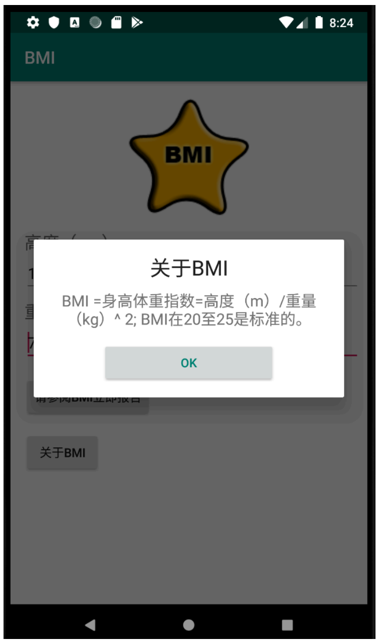
重量（kg）

70


BMI

您的BMI值为 21.85

你看起来很棒!


关于BMI

BMI =身高体重指数=高度（m）/重量（kg）＾2; BMI在20至25是标准的。

Check Point 1:

1. Chinese Localization of the BMI App

Instructor Verification (separate page)

1.2. Supporting Different Screens

Android categorizes device screens using two general properties: **size** and **density**. You should expect that your app will be installed on devices with screens that range in both size and density. As such, you should include some alternative resources that optimize your app’s appearance for different screen sizes and densities.

- There are four generalized sizes: small, normal, large, xlarge
- And five generalized densities:
    - mdpi
    - hdpi
    - xhdpi
    - xxhdpi
    - xxxhdpi

One of the difficulties developers had in pre-3.2 Android devices was the "large" screen size bin, which encompasses the Dell Streak, the original Galaxy Tab, and 7" tablets in general. However, many applications may want to show different layouts for different devices in this category (such as for 5" and 7" devices), even though they are all considered to be "large" screens. That's why Android introduced the **"Smallest-width” qualifier** (amongst others) in Android 3.2.

The Smallest-width qualifier allows you to target screens that have a certain minimum width given in dp. For example, the typical 7" tablet has a minimum width of 600 dp, so if you want your UI to have two panes on those screens (but a single list on smaller screens), you can use the same two layouts from the previous section for single and two-pane layouts, but instead of the large size qualifier, use sw600dp to indicate the two-pane layout is for screens on which the smallest-width is 600 dp:

- sw600dp

To declare different layouts and bitmaps you'd like to use for different screens, you must place these alternative resources in separate directories, similar to how you do for different language strings.

Also be aware that the screens orientation (landscape or portrait) is considered a variation of screen size, so many apps should revise the layout to optimize the user experience in each orientation.

1.2.1 Create Different Layouts

To optimize your user experience on different screen sizes, you should create a unique layout XML file for each screen size you want to support. Each layout should be saved into the appropriate resources directory, named with a **`-<screen_size>`** suffix. For example, a unique layout for large screens should be saved under **`res/layout-sw600dp/`**. For example, this project includes a default layout and an alternative layout for *large* screens:

```
MyProject/
     res/
         layout/
             activity_main.xml
         layout-sw600dp/
             activity_main.xml
```

The file names must be exactly the same, but their contents are different in order to provide an optimized UI for the corresponding screen size.

Simply reference the layout file in your app as usual:

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

The system loads the layout file from the appropriate layout directory based on screen size of the device on which your app is running. More information about how Android selects the appropriate resource is available in the Providing Resources guide.

As another example, here's a project with an alternative layout for landscape orientation:

```
MyProject/
     res/
         layout/
             activity_main.xml
         layout-land/
             activity_main.xml
```

By default, the `layout/activity_main.xml` file is used for portrait orientation.

If you want to provide a special layout for landscape, including while on large screens, then you need to use both the `sw600dp` and `land` qualifier:

```
MyProject/
     res/
         layout/              # default (portrait)
```

```
            activity_main.xml
        layout-land/         # landscape
            activity_main.xml
        layout-sw600dp/        # smallest-width 600dp (portrait)
            activity_main.xml
        layout-sw600dp-land/   # smallest-width 600dp landscape
            activity_main.xml
```

Android 3.2 and above supports an advanced method of defining screen sizes that allows you to specify resources for screen sizes based on the minimum width and height in terms of density-independent pixels. This lesson does not cover this new technique. For more information, read Designing for Multiple Screens.

1.2.2 Create Different Bitmaps

You should always provide bitmap resources that are properly scaled to each of the generalized density buckets: low, medium, high and extra-high density. This helps you achieve good graphical quality and performance on all screen densities.

To generate these images, you should start with your raw resource in vector format and generate the images for each density using the following size scale:

- xhdpi: 2.0
- hdpi: 1.5
- mdpi: 1.0 (baseline)
- ldpi: 0.75

This means that if you generate a 200x200 image for xhdpi devices, you should generate the same resource in 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi devices.

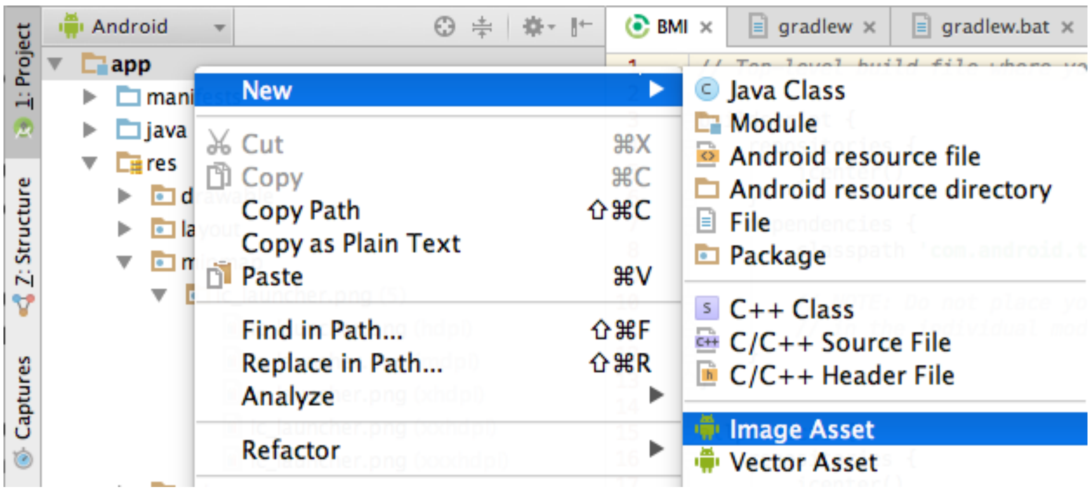
Then, place the files in the appropriate drawable resource directory:

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

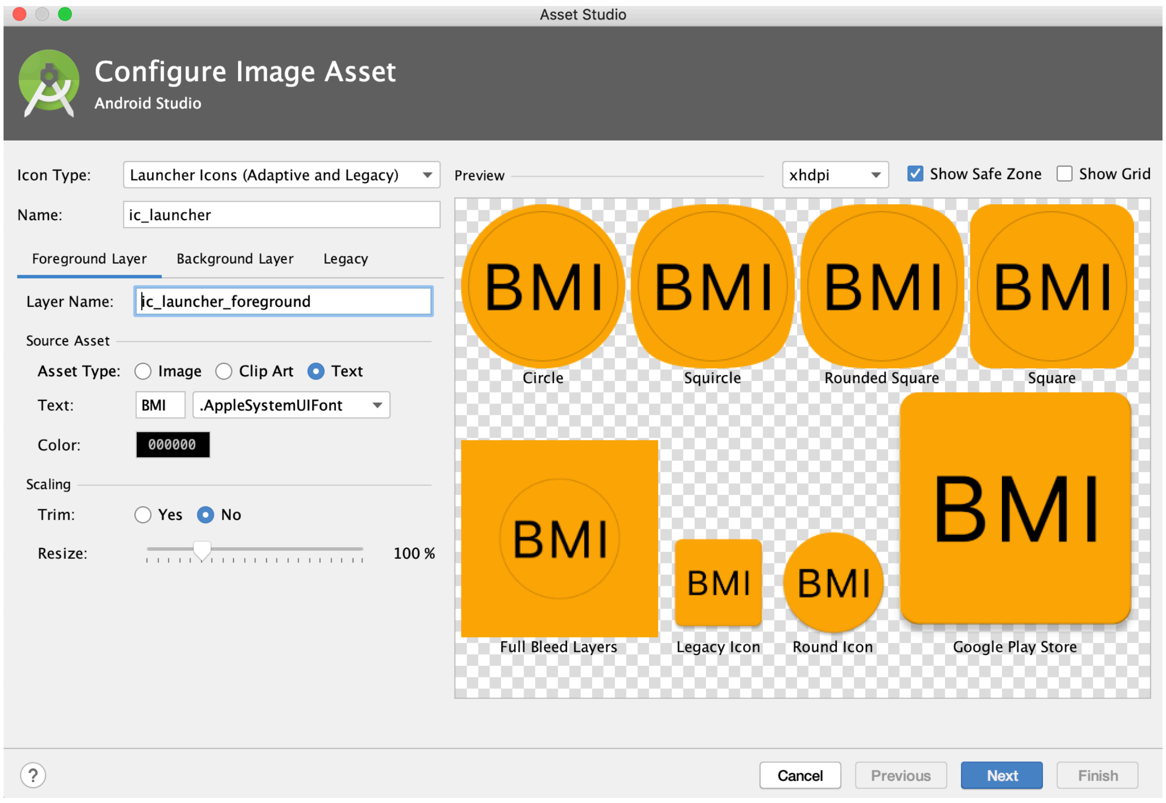
Any time you reference `@drawable/awesomeimage`, the system selects the appropriate bitmap based on the screen's density.

1.2.3 Create Different Launch Icon using Android Studio Image Asset

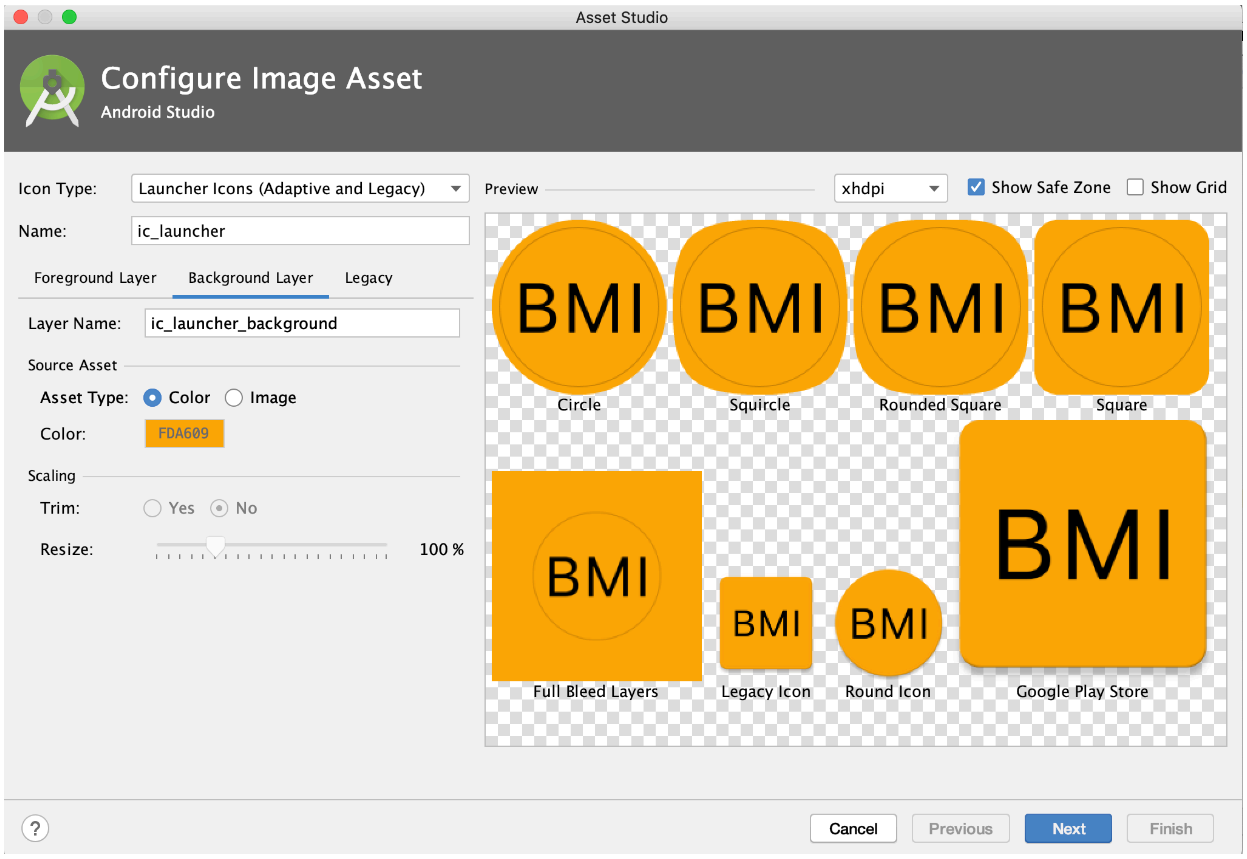
In the Android Studio, a very power image tool, which is called Image Asset, is provided for creating image files in different resolutions. In this exercise, students are required to use this Image Asset tool to create a new “launcher icon” for the BMI app. Open your BMI project again and then right click your “app” directory at the Android project view with selection of “New => Image Asset” as shown below:



In the Asset Studio window, you can use “Image”, “Clipart” and “Text” as Foreground to create the Launcher icons. In this exercise, students are required to use the “Text” Foreground with the text of “BMI” in the “Foreground Layer” and then adjust the “Background Layer” to create a new icon with different resolutions as shown below:



Icon Type: Launcher Icons (Adaptive and Legacy)
Name: ic_launcher
Layer Name: ic_launcher_foreground
Asset Type: Image Clip Art [✓ Text]
Text: BMI .AppleSystemUIFont
Color: 000000
Trim: Yes [✓ No]
Resize: 100 %
Preview: xhdpi [✓ Show Safe Zone] Show Grid


Asset Studio

Configure Image Asset
Android Studio

Icon Type: Launcher Icons (Adaptive and Legacy)
Name: ic_launcher

Foreground Layer | Background Layer | Legacy

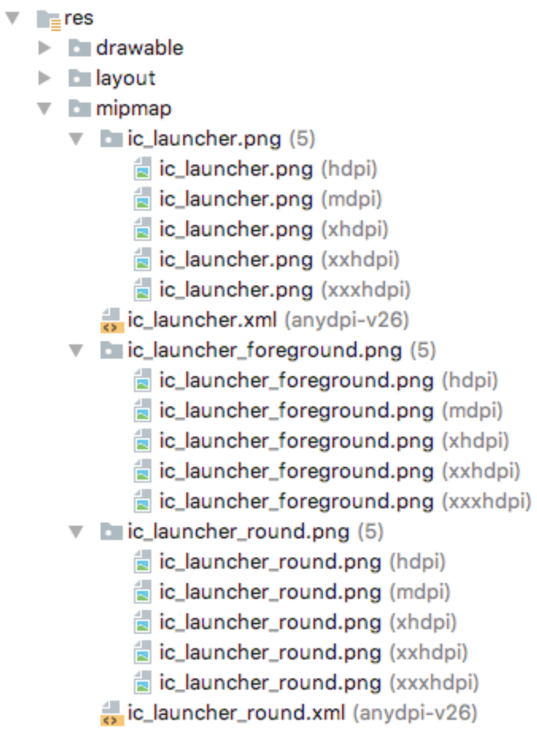
Layer Name: ic_launcher_background

Source Asset
Asset Type: [✓ Color] Image
Color: FDA609

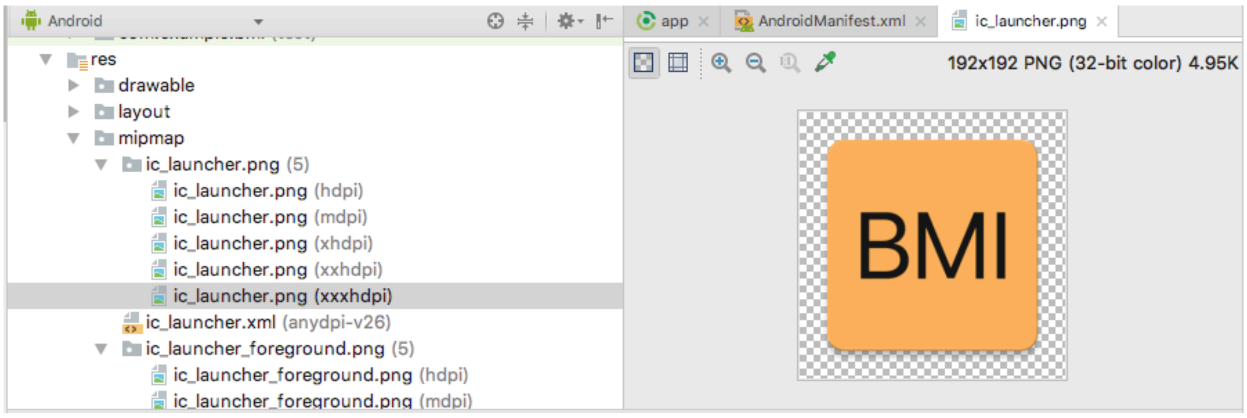
Scaling
Trim: Yes [✓ No]
Resize: 100 %

Preview: xhdpi [✓ Show Safe Zone] Show Grid

Click "Next" and then "Final" to generate these image files with filename of "ic_launcher.png in the different folders of "mipmap" as shown below:

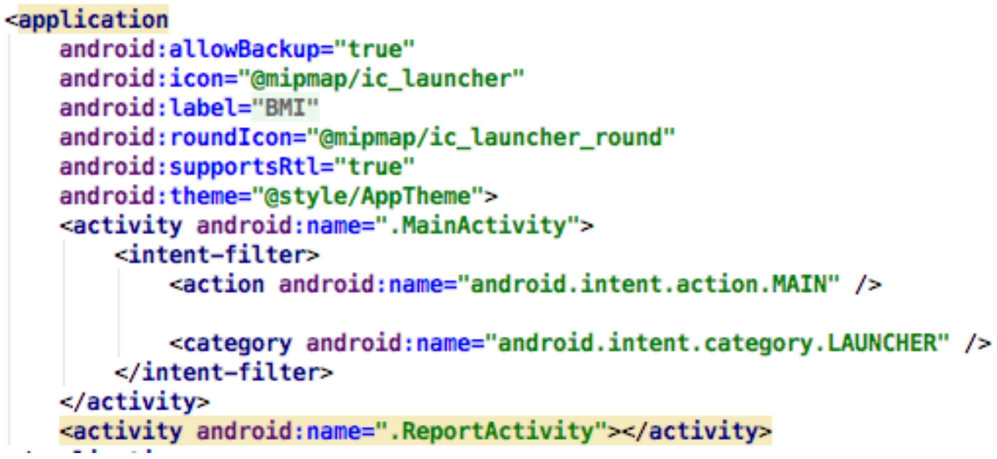

```
res
  drawable
  layout
  mipmap
    ic_launcher.png (5)
      ic_launcher.png (hdpi)
      ic_launcher.png (mdpi)
      ic_launcher.png (xhdpi)
      ic_launcher.png (xxhdpi)
      ic_launcher.png (xxxhdpi)
    ic_launcher.xml (anydpi-v26)
    ic_launcher_foreground.png (5)
      ic_launcher_foreground.png (hdpi)
      ic_launcher_foreground.png (mdpi)
      ic_launcher_foreground.png (xhdpi)
      ic_launcher_foreground.png (xxhdpi)
      ic_launcher_foreground.png (xxxhdpi)
    ic_launcher_round.png (5)
      ic_launcher_round.png (hdpi)
      ic_launcher_round.png (mdpi)
      ic_launcher_round.png (xhdpi)
      ic_launcher_round.png (xxhdpi)
      ic_launcher_round.png (xxxhdpi)
    ic_launcher_round.xml (anydpi-v26)
```

These image files are specified for different screen density with sizes of 48x48 (mdip), 72x72 (hdip), 96x96 (xhdip), 144x144 (xxhdip), and 192x192 (xxxhdip).



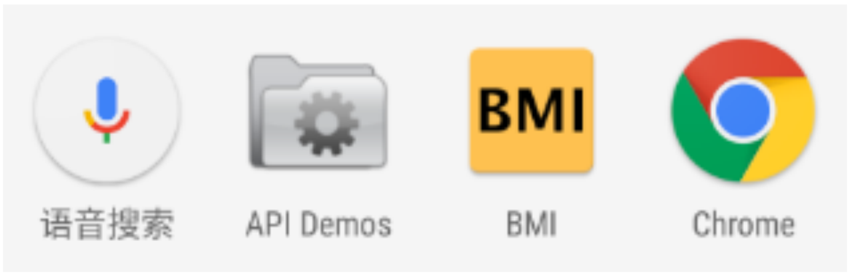
Android
app
AndroidManifest.xml
ic_launcher.png
res
drawable
layout
mipmap
ic_launcher.png (5)
ic_launcher.png (hdpi)
ic_launcher.png (mdpi)
ic_launcher.png (xhdpi)
ic_launcher.png (xxhdpi)
ic_launcher.png (xxxhdpi)
ic_launcher.xml (anydpi-v26)
ic_launcher_foreground.png (5)
ic_launcher_foreground.png (hdpi)
ic_launcher_foreground.png (mdpi)
192x192 PNG (32-bit color) 4.95K
BMI

To use these image files as launcher icon for your app, you have to specify this resource in the AndroidManifest.xml with `android:icon="mipmap/ic_launcher"` and `android:roundIcon="mipmap/ic_launcher_round"` as shown below:



```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="BMI"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

Run your BMI again and your new BMI launcher icon will be changed as shown bellow:

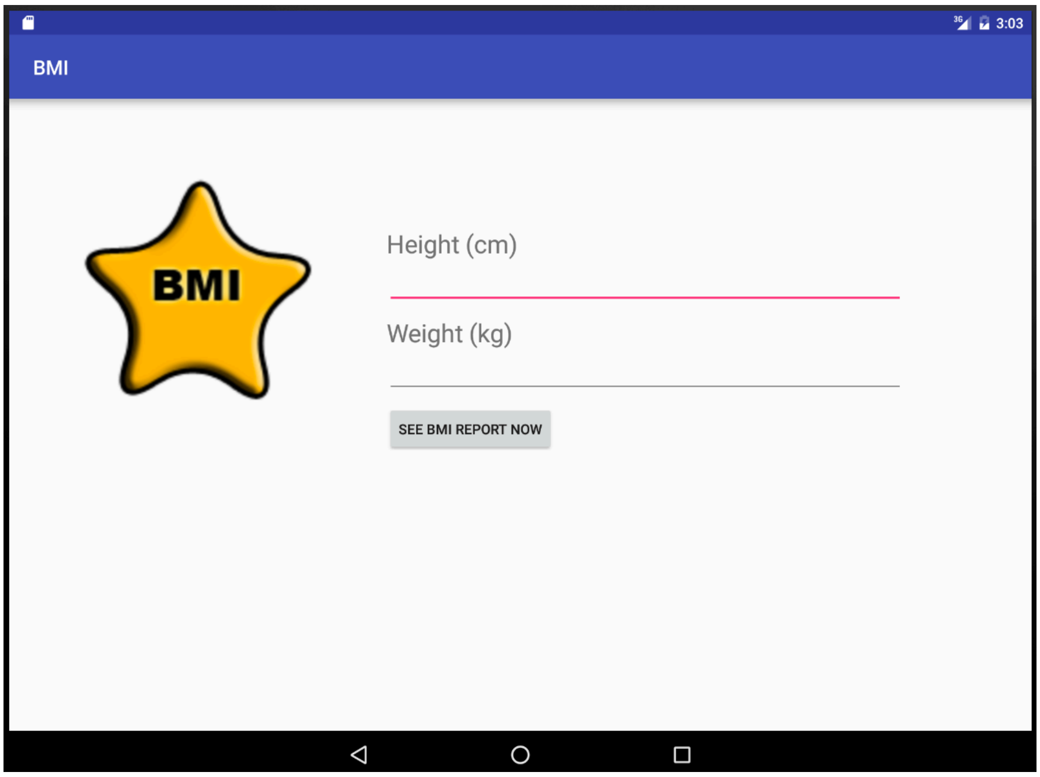


Check Point 2:

2. Create New Launch Icon with multiple resolutions for BMI app.
Instructor Verification (separate page)

1.2.3 Enhance BMI Project with Support of Large Screen Size (Optional)

Further enhance the BMI project with basic features of the Lab02 and make it supports Nexus 10 devices with 10.055 inches screen and 2560x1600 resolution (xhdpi) using a folder of "`res/layout-sw600dp-land`", in which modify the layout_height, layout_width of the ImageView and textSize with 30sp for the TextView objects. Create a Nexus 10 AVD to run your completed project and evaluate the UIs.



II. Android Menus and Action Bar

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an action bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This section shows how to create the three fundamental types of menus or action presentations on all versions of Android:

Options Menu and Action Bar

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

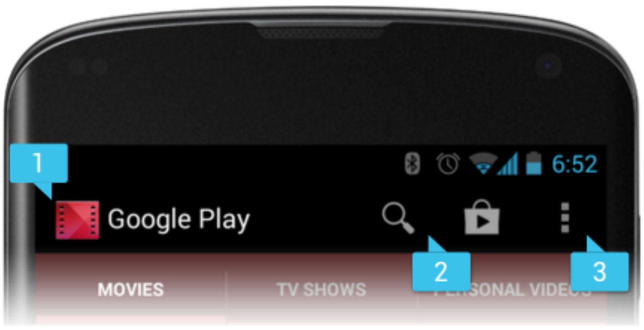
If you're developing for Android 2.3 or lower, users can reveal the options menu panel by pressing the *Menu* button.

On Android 3.0 and higher, items from the options menu are presented by the action bar as a combination of on-screen action items and overflow options. Beginning with Android 3.0, the *Menu* button is deprecated (some devices don't have one), so you should migrate toward using the action bar to provide access to actions and other options.

Context Menu and Contextual Action Mode

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

When developing for Android 3.0 and higher, you should instead use the contextual action mode to enable actions on selected content. This mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

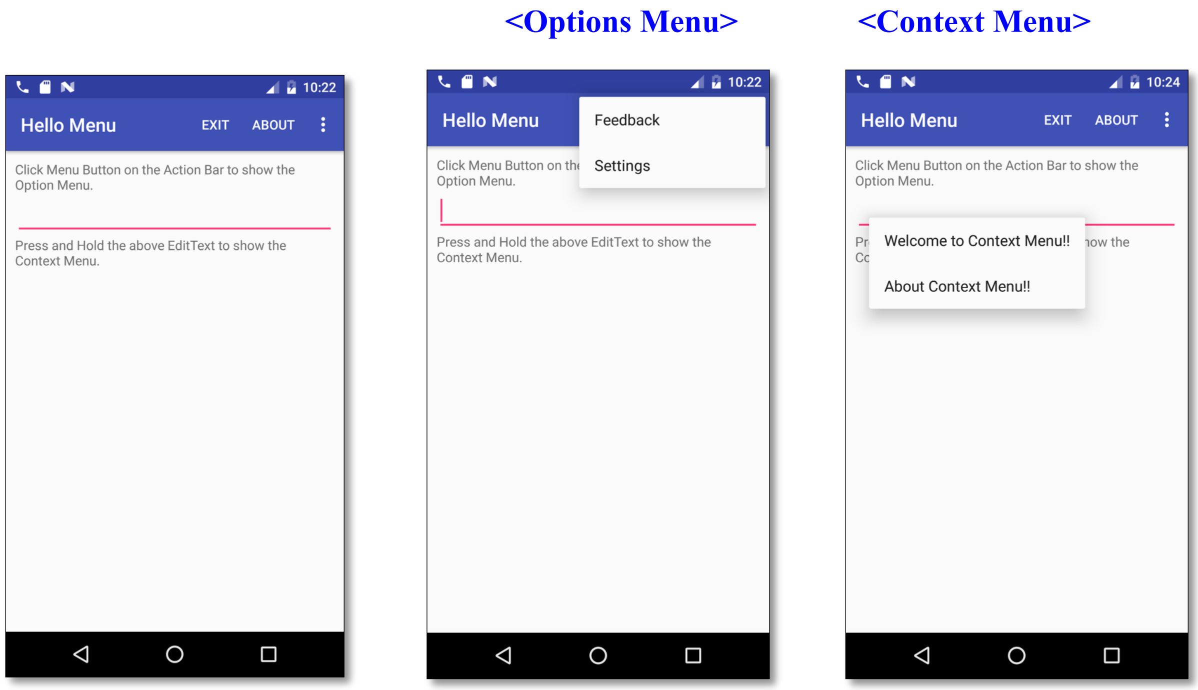


Popup menu

A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

2.2. Create Android Menus Projects

In this section, students are required to create an application that provides an **Options Menu** on the Action Bar and a **Context Menu** on the EditText as shown below:


<Options Menu>
<Context Menu>
10:22
Hello Menu
EXIT
ABOUT
Click Menu Button on the Action Bar to show the Option Menu.
Press and Hold the above EditText to show the Context Menu.
10:22
Hello Menu
Feedback
Settings
10:24
Hello Menu
EXIT
ABOUT
Welcome to Context Menu!!
About Context Menu!!

To display the Options Menu, press the menu button on the top right side of the action bar. In the Hello Menu app, the option menu consists of four items (Settings, About, Feedback and Exit). In this app, the context menu can be displayed by pressing and hold the EditText and the Context Menu consists of two items (Welcome to Context Menu! and About Context Menu!).

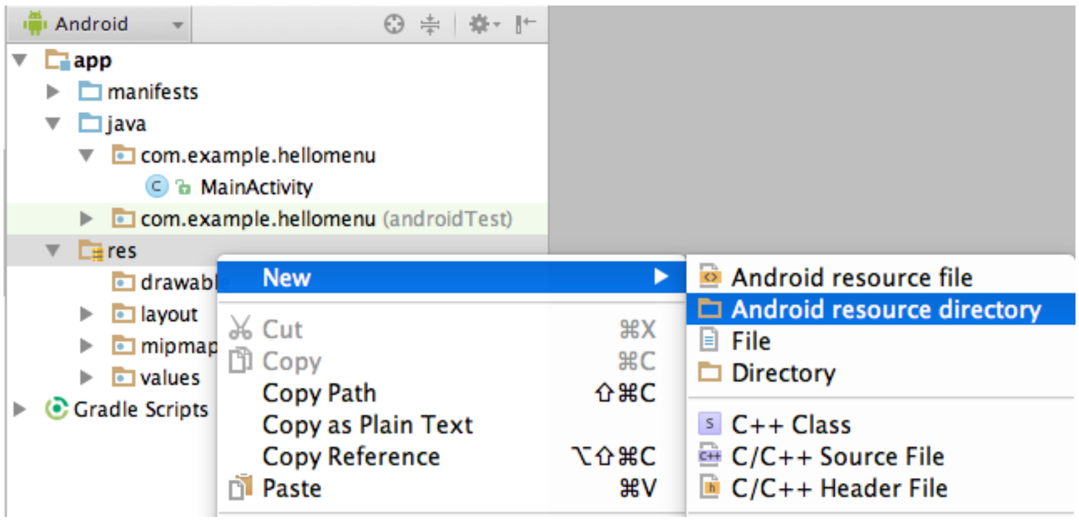
1. Create a project with application name of "**Menus**" with an Empty activity.
2. To define some external values in **`res/values/strings.xml`**, modify its content as listed below:

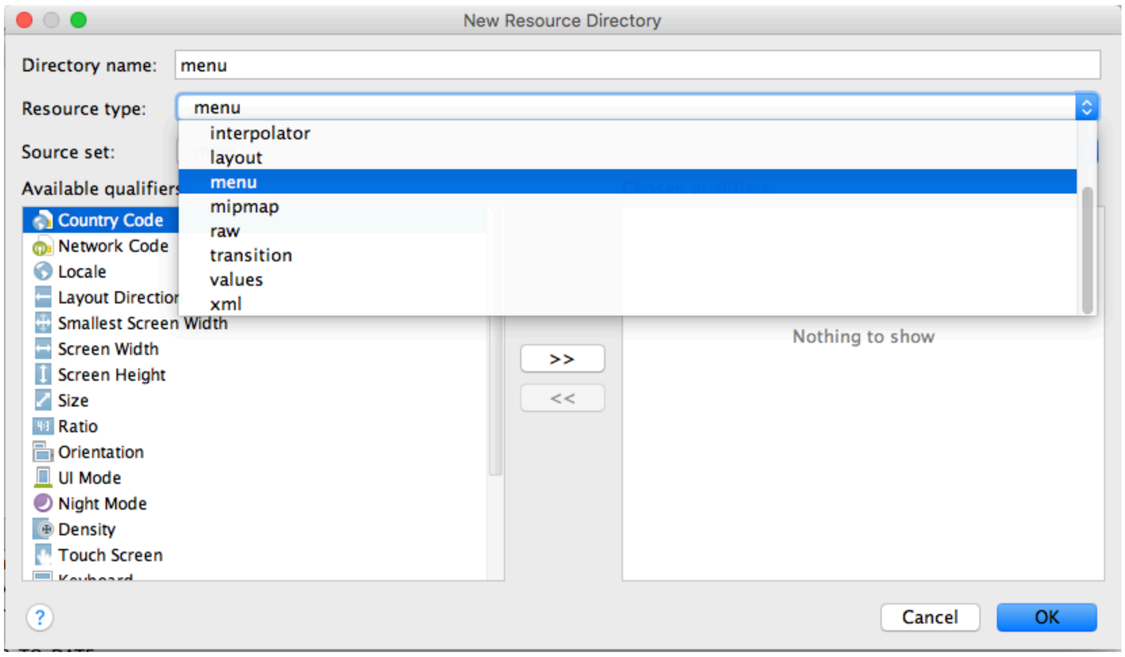
```
<resources>
    <string name="app_name">Menus</string>

    <string name="menu_settings">Settings</string>
    <string name="about">About</string>
    <string name="feedback">Feedback</string>
    <string name="exit">Exit</string>

    <string name="textView1">Click Menu Button on the Action Bar to show the Option
Menu. </string>
    <string name="textView2">Press and Hold the above EditText to show the Context
Menu. </string>

    <string name="welcome_msg">Welcome to Context Menu!!</string>
    <string name="about_msg">About Context Menu!!</string>

</resources>
```

3. To create the layout of the activity in **`res/layout/activity_main.xml`**, modify its content as listed below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="15dp" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/textView1" />
    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <requestFocus />
    </EditText>
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textView2" />
</LinearLayout>
```

4. To create a **`res/menu`** folder and then create two xml files for the Options Menu and Context Menu with filenames of **`menu_options.xml`** and **`menu_context.xml`** under the **`res/menu`** folder. To create the menu folder, you can right click the res folder in the Android view and select **New->Android resource directory** and then select **menu Resource type** in the pop-up window as shown below:




New Resource Directory

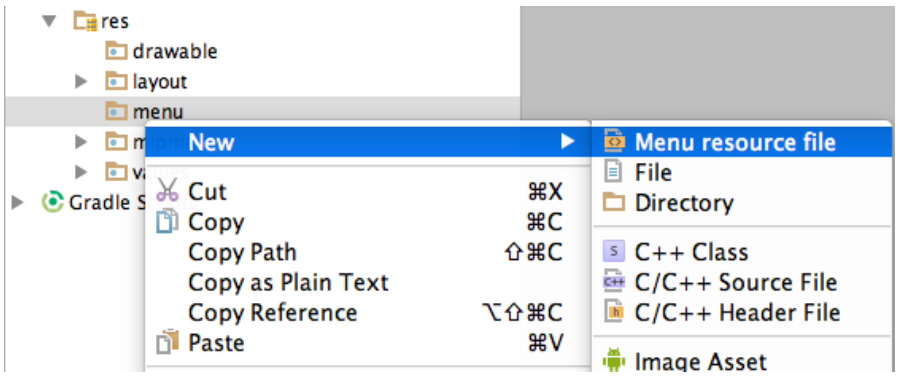
Directory name: menu

Resource type: menu

Source set:

Available qualifiers:

After creating the res/menu folder, you can right click this folder and select **New->Menu resource file** to create a menu_options.xml file as shown below:



5. The Options menu's items are defined in the **`menu_options.xml`** file and its content as listed below:

```
<?xml version="1.0" encoding="utf-8"?>
<menu 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"
    tools:context=".MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/menu_settings"
        app:showAsAction="never" />
    <item
        android:id="@+id/menu_about"
        android:title="@string/about"
        android:orderInCategory="1"
        app:showAsAction="always" />

    <item
        android:id="@+id/menu_feedback"
        android:title="@string/feedback" />
    <item
        android:id="@+id/menu_exit"
        android:title="@string/exit"
        app:showAsAction="ifRoom" />
</menu>
```

In this options menu, the **Settings** is used the "never" in the `showAsAction` attribute such that it will never be shown on the action bar but the **About** item is used "`always`" attribute such that it is always appeared on the action bar. While the **Exit** item is used the "`ifRoom`" attribute such that it will depend on the space available on the action bar if the space is available then it will be displayed.

6. The Context Menu's items is defined in the **`menu_context.xml`** file and its content as listed below:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/menuItemWelcome"
        android:title="@string/welcome_msg" />
    <item
        android:id="@+id/menuItemAbout"
        android:title="@string/about_msg" />
</menu>
```

7. Inflating a menu resource (`menu_options.xml`) by `adding onCreateOptionsMenu(Menu menu)` in the main Activity. Menu items in `menu_main.xml` will appear when the user touches the MENU button. Response to menu click events by overriding `onOptionsItemSelected(Menu menu)` in the MainActivity. Add the following codes into the `MainActivity.java` class.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the aaction bar if it is present
    getMenuInflater().inflate(R.menu.menu_options, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {

    int id = item.getItemId();

    switch (id) {
        case R.id.action_settings:
            Toast.makeText(this, "Settings Button Clicked !",
                    Toast.LENGTH_LONG).show();
            return true;
        case R.id.menu_about:
            Toast.makeText(this, "About Button Clicked !",
                    Toast.LENGTH_LONG).show();
            return true;
        case R.id.menu_feedback:
            Toast.makeText(this, "Send Feedback Button Clicked !",
                    Toast.LENGTH_LONG).show();
            return true;
        case R.id.menu_exit:
            finish();
            return true;
    }
    return false;
}
```

8. By calling `registerForContextMenu()` and passing it a View (an EditText in this example) you assign it a context menu. When this View (EditText) receives a long-press, it displays a context menu. Add the following codes into the `MainActivity.java` class.

```
public class MainActivity extends AppCompatActivity {

    private EditText mOutEditText;

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

        mOutEditText = (EditText) findViewById(R.id.editText);
        registerForContextMenu(mOutEditText);
    }
        :
        :
```

9. By overriding the activity's context menu create callback method, **`onCreateContextMenu()`**. Add the following codes into the `MainActivity.java` class.

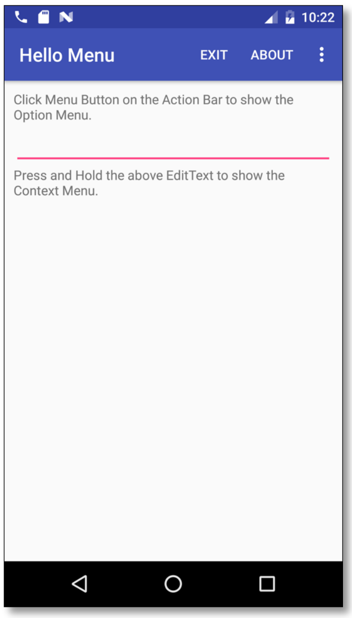
```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {

    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
}
```

10. By overriding your activity's menu selection callback method for context menu, **`onContextItemSelected()`**. Add the following codes into the MainActivity.java class.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menuItemWelcome:
            mOutEditText.setText( this.getResources().getText( R.string.welcome_msg)
);
            return true;
        case R.id.menuItemAbout:
            mOutEditText.setText( this.getResources().getText( R.string.about_msg) );
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

11. Run your completed project and demonstrate these option and context menus with toast messages.

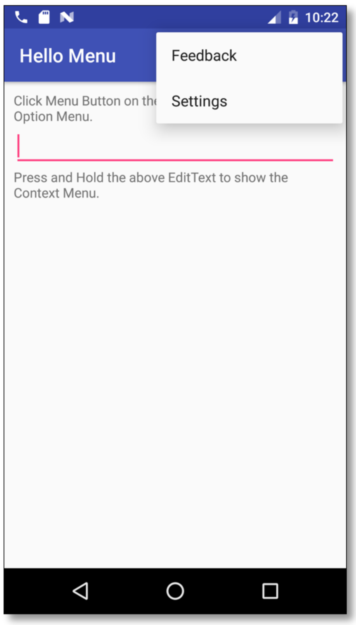
Exit and About Item of Option Menu is on Action Bar



Click Menu Button on the Action Bar to show the Option Menu.

Press and Hold the above EditText to show the Context Menu.

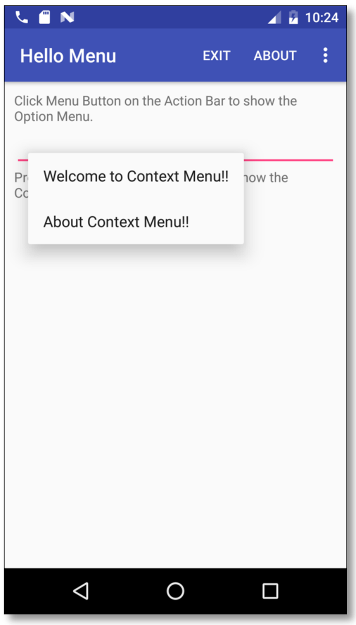
After Press Option Menu Button, the other two items will be displayed



Click Menu Button on the Action Bar to show the Option Menu.

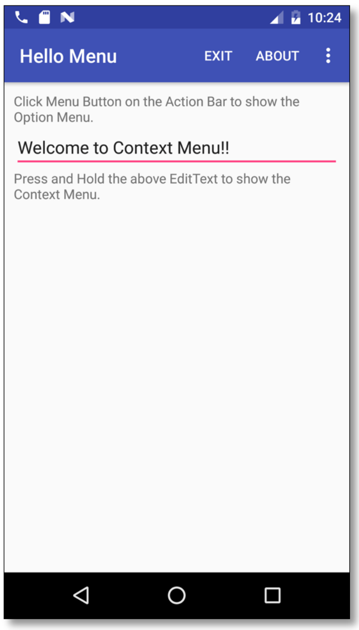
Press and Hold the above EditText to show the Context Menu.

Long Press on the EditText field, then the Context Menu will be appeared



Click Menu Button on the Action Bar to show the Option Menu.

After Press on the first Item of the Context Menu, Welcome message is added



Click Menu Button on the Action Bar to show the Option Menu.

Welcome to Context Menu!!

Press and Hold the above EditText to show the Context Menu.

Check Point 3:

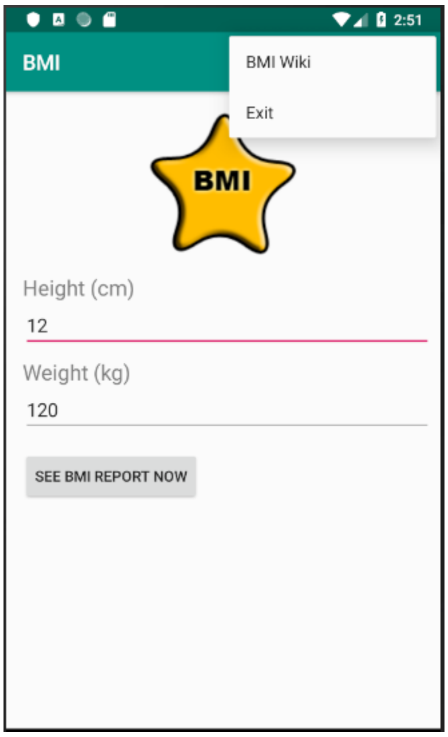
3.1 Options Menu for Menus project 3.1 Context Menu for Menus project
Instructor Verification (separate page)

2.3 Adding Options and Context Menus for BMI App

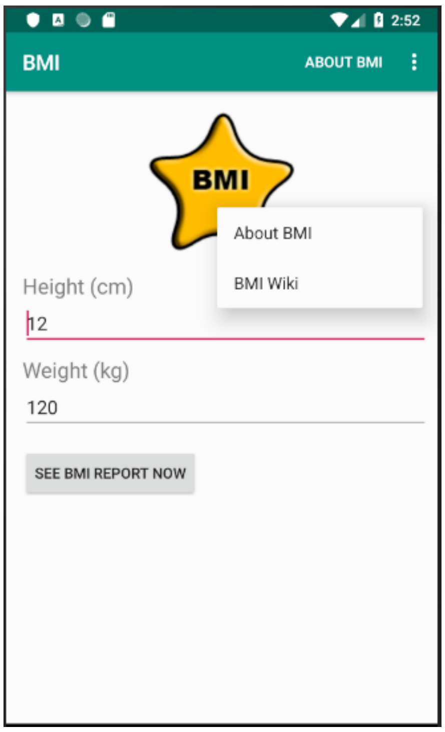
1. Based on the “Menus” app illustrated in the previous section to enhance the BMI App of Lab03 with “About BMI”, “BMI Wiki” and “Exit” options in Options Menu and a Context Menu on long press of the BMI logo to show the “About BMI” and “BMI Wiki”.

- About BMI : Display a dialog for showing the “About the BMI” message as shown in lab03.
- BMI Wiki : Open a the BMI Wiki webpage with URL of http://en.wikipedia.org/wiki/Body_mass_index by implicit intents as shown in lab4.
- Quit: Exit the current activity using the `finish()` method.

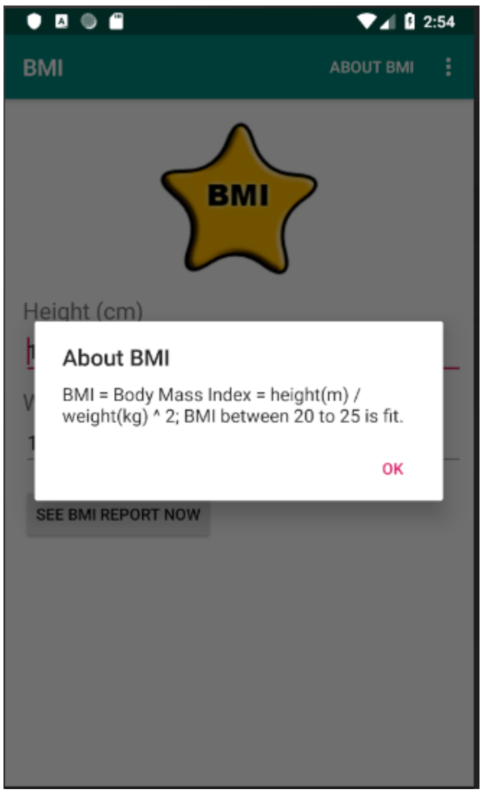
2. Run the application again. Use the option menu’s About BMI and context menu’s About BMI to display the “About BMI” message as shown below:



Option Menu



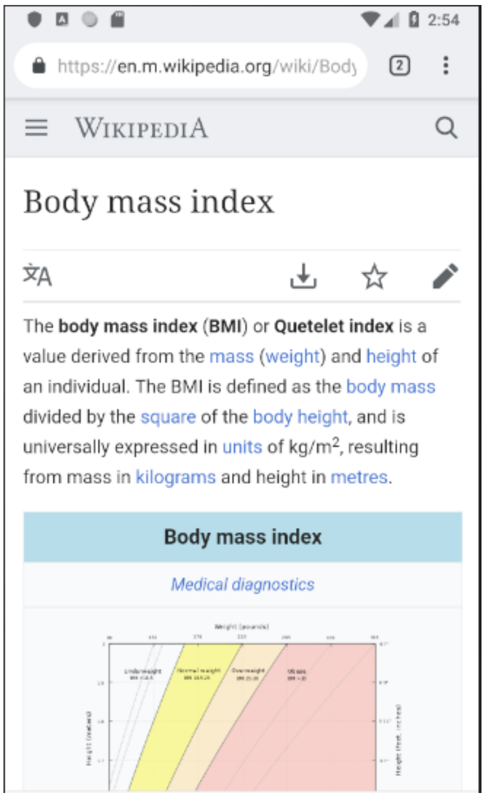
Context Menu
(Long Press of Image View)



About BMI

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

About BMI



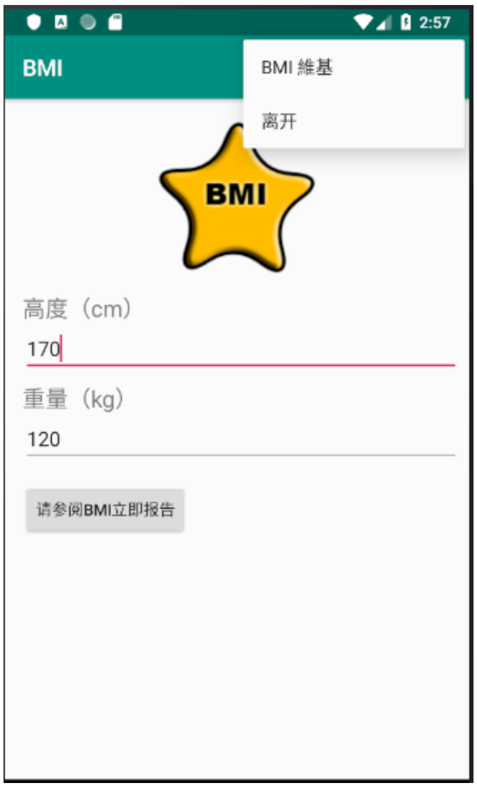
Body mass index

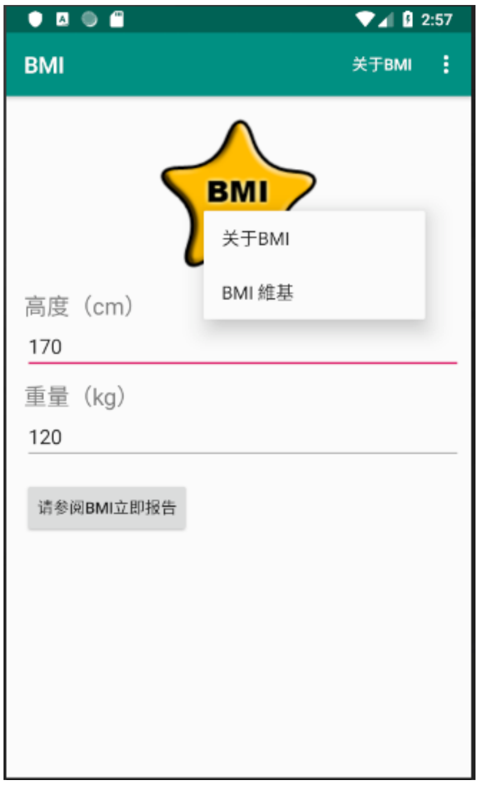
The **body mass index** (**BMI**) or **Quetelet index** is a value derived from the mass (weight) and height of an individual. The BMI is defined as the body mass divided by the square of the body height, and is universally expressed in units of kg/m^2, resulting from mass in kilograms and height in metres.

Body mass index

BMI WiKi

3. Modify the option menu implementation for making it support both English and Chinese.





4. Suggested `menu_options.xml` and code for implementing the `onCreateOptionsMenu()` and `onOptionsItemSelected()` methods are given as below.

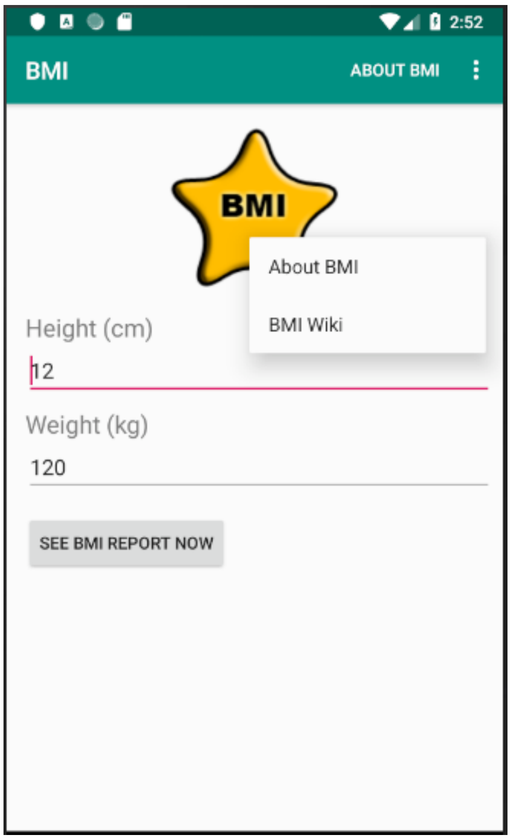
`res/menu/menu_options.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" >
    <item
        android:id="@+id/menu_about"
        app:showAsAction="always"
        android:title="@string/menu_about" />
    <item
        android:id="@+id/menu_wiki"
        android:title="@string/menu_wiki" />
    <item
        android:id="@+id/menu_exit"
        android:title="@string/menu_exit" />
</menu>
```

Code for implementing options menu in **`MainActivity.java`**

```
// --- Option Menu ---
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_options, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {
        case R.id.menu_about:
            openOptionsDialog();
            return true;

        case R.id.menu_wiki:
            Intent intent = new Intent(Intent.ACTION_VIEW,
                    Uri.parse("http://en.wikipedia.org/wiki/Body_mass_index"));
            startActivity(intent);
            return true;

        case R.id.menu_exit:
            finish();
            return true;
    }
    return false;
}

public void openOptionsDialog() {
    new AlertDialog.Builder(MainActivity.this)
            .setTitle(R.string.menu_about)
            .setMessage(R.string.about_msg)
            .setPositiveButton(R.string.ok,
                    new DialogInterface.OnClickListener() {
                        public void onClick(
                                DialogInterface dialoginterface, int i) {
                        }
                    }).show();
}
```

Students are also required to implement the `menu_context.xml` file and modify the `MainActivity.java` code for context menu with long press on the Image View as shown below:



Check Point 4:

4.1 Option Menu for BMI project 4.1 Context Menu for BMI project
Instructor Verification (separate page)