Learning Objectives:

1. Design and write an app that has a screen that **allows the user to enter data and calculates a person's BMI (Body Mass Index).**
2. Use ViewModel to keep track of the UI data.

Requirements

Part 1: Activity

For the **main** screen of your app, your app needs to satisfy the below the requirements:

1. Users should be able to enter their **weight** and **height** in English units (pounds and inches) and then calculate and display the **BMI**. For the height, you may design to have one single field to allow the user to input height in inches. Or you may have two fields: one field for feet and the other for inches for the user to input the person's height data.

   Your calculation should be based on the below:

   BMI formula

   $$\text{BMI} = \frac{\text{weight in pounds} * 703}{\text{height in inches} * \text{height in inches}}$$

2. The app should also display a message based on the following information:

BMI less than 18.5	Underweight
BMI between 18.5 and 24.9	Normal
BMI greater than 24.9 and less than 30	Overweight
BMI greater than or equal to 30	Obese

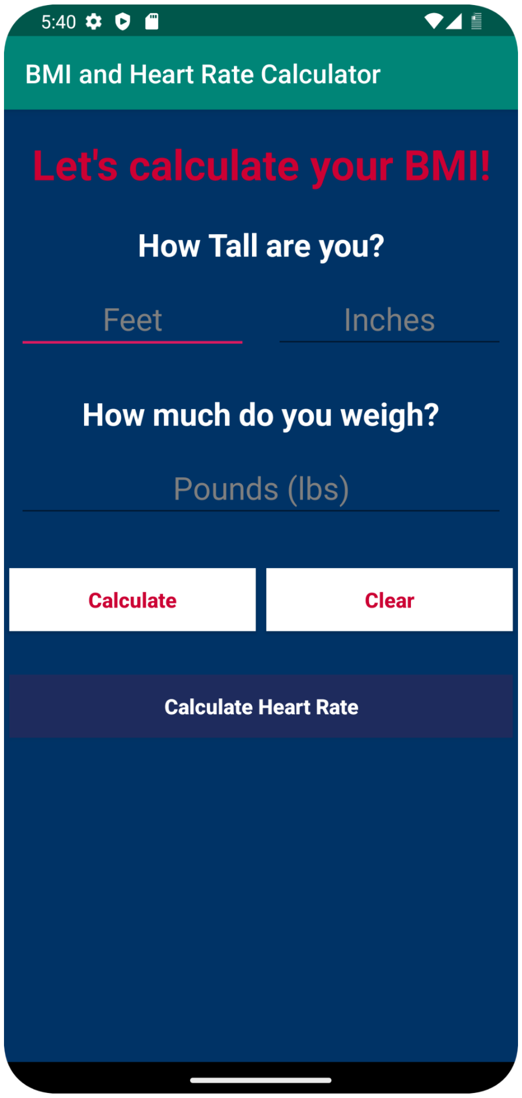
3. Your app needs to have ***three*** buttons. First, your app should be able to perform the calculation when the user clicks the **Calculate** button. Second, when a user clicks the button **Clear,** your app would clear the text input and display boxes. Third, the button **Calculate Heart**

Rate is presented in your app UI but it does nothing when the user clicks the button in this assignment. That means, if the user clicks it, there is no action. In the following assignment, you will extend this assignment to start the second activity with the third button.

You should design a layout for the main screen to format the label/entry pairs and you should have a text box on the screen (following other widgets) to display the calculated information("Normal" or "Overweight" or etc) or inform errors like missing data.

The text boxes for input should allow only numbers, with a decimal point if you think that is appropriate. (You can use the EditText widget for text input.) The numeric output should be formatted to display one digit to the right of the decimal point.

The initial screen could be like below:



Note that your app is NOT required to have the exact UI like the above one. You just need to make sure you have the required UI elements as well as actions performed when the user interacts with your app.

I also want to point out that the default base theme provided in Android Studio when you create a new project has no action bar. If you want to have an action bar created with your app like the

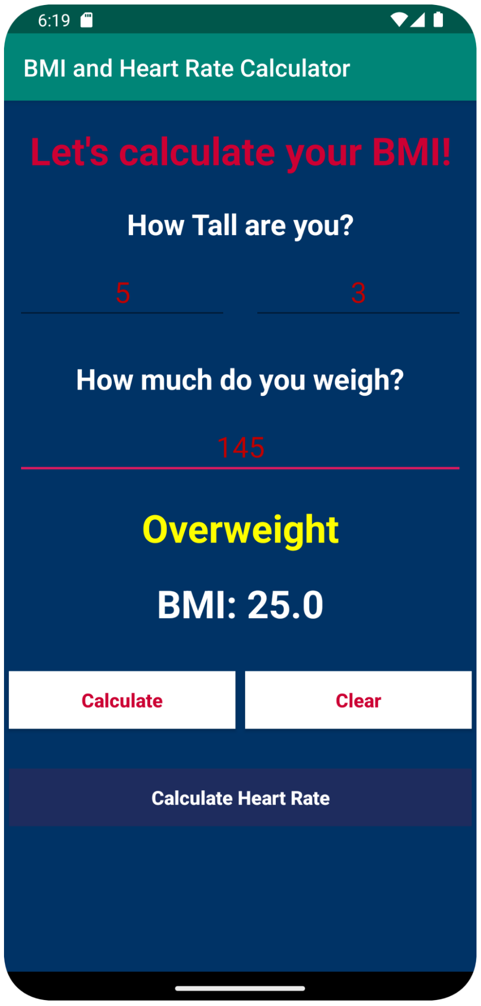
UI I use here, you could use a different base theme parent, which I highlight below, to have an action bar. The styles can be revised in the **themes.xml** in the **res/values** folder. We will cover the action bar topic later. I also include the three items I use in my UI theme. The colors are defined in a similar way to how they are defined in the demo project GeoQuiz.

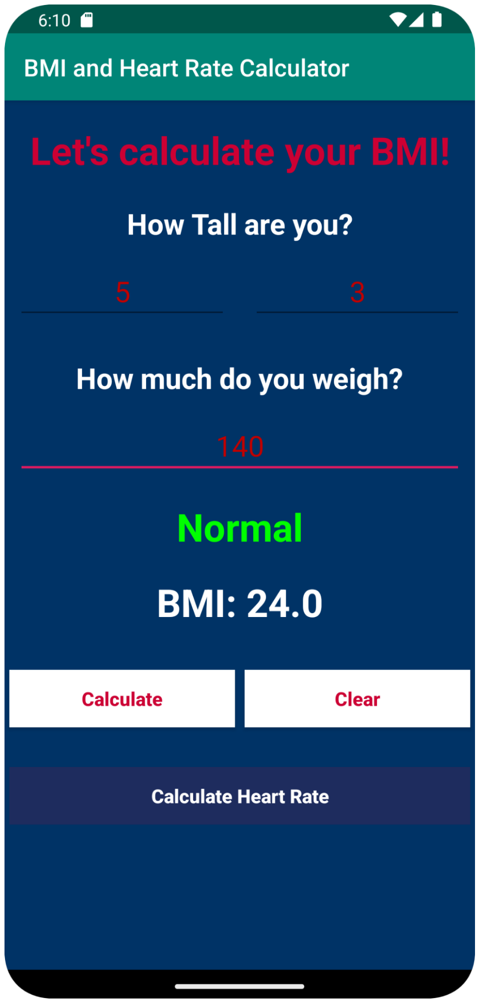
```
<style name="Base.Theme.yourappname"
parent="Theme.AppCompat.Light.DarkActionBar">
   <!-- Customize your theme here. -->
   <item name="colorPrimary">@color/colorPrimary</item>
   <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
   <item name="colorAccent">@color/colorAccent</item>

</style>
```

For the night theme file, you can also change the parent into

```
<style name="Base.Theme.yourappname"
parent="Theme.AppCompat.DayNight.DarkActionBar">
 …whatever here
</style>
```

After the user makes input on height and weight and clicks the Calculate button, the BMI result gets displayed.





Part 2: ViewModel

Additionally, you need to add a ViewModel class to keep track of the UI data so that your ViewModel is able to work with your main activity. This part intends to provide you an opportunity for you to practice creating ViewModel objects.

1. Your ViewModel class should be able to maintain the UI data for the BMI app's main activity. It is also used to **keep the decision making logic such as conducting the BMI calculation.**
2. You need to use different, consistent colors to reflect the calculated BMI categorical information. For example, for either "underweight" or "overweight", your app presents it using yellow. Your app uses green color to present "normal" and red to present "obese".
3. When running the app, the UI data including weight, height, and calculated BMI data keeps the same even after rotating the device.

Lastly, use your own good judgment in designing the app's layout, but make sure everything is labeled and easy to understand. If you aren't sure, ask.

Notes on Numeric Output Formatting

Floating-point numeric output in a text box or log will display decimal places - but you can't control how many without some work. One technique to control the number of decimal places displayed is by making use of a special Java class. Do the following:

```
import java.text.DecimalFormat
```

Create a DecimalFormat object as follows:

```
 val df: DecimalFormat = DecimalFormat("#.0") // for 1 decimal places
```

When you want to display a number, pass it to the *format()* method of this object:

```
"Number is " + df.format(adouble)  // or anything that wants a String
```

The *format()* method takes a double value and returns a String object, which is the text representation of the double's value, formatted as specified in the constructor of the *DecimalFormat* object.

Notes on build.gradle.kts (module: app)

Due to the gradle updates in Android Studio you are using, you need to adjust the syntax you use, which could be a bit different from the textbook projects, when building your list of dependencies and other configurations in your build.gradle.kts(module:app).   For example, below should be the two libraries you add to dependencies:

```
dependencies {
…
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.1")
implementation("androidx.activity:activity-ktx:1.7.2")
…
}
```

In the android section, you also need to add the below highlighted to use viewmodels in your app.

```
android {
…
    buildFeatures {
        viewBinding = true
    }
…
}
```

Notes on how to approach this assignment

The two parts should be resolved one by one. For part 1, you should work it out first. For this part 1 task, you can also divide it into several parts—building the activity code and the resources.   Make sure you check and test each component you add to the app project incrementally. It is a bad idea to just write code and test all of the components.

Turn-in

Turn in your application package to Assignment Dropbox folder Assignment 2 BMI. You should turn in everything included in your app to the dropbox. You can use the File menu of Android Studio and choose Export to Zip. And then send the zip package to the assignment folder.