CSCI3310 Mobile Computing & Application Development

Lab 01 – Basic UI: Magic Number

Objective

In this lab, you learn how to create your first interactive app using the **Empty Activity** template. You also learn how to use the layout editor to design a layout, and how to edit the layout in XML. Steps:

1) Create an app and add `Button` elements and `TextView` elements to the layout.
2) Manipulate each element in the `ConstraintLayout` to constrain them to the margins and other elements.
3) Change UI element attributes and edit the app's layout in XML.
4) Implement click-handler methods to display messages on the screen when the user taps each `Button`.

Magic Number Guessing

MagicNumber app is a simple number guessing game, where the user can guess a magic number a maximum of 3 times. The app consists of three `Button` elements (+), (-) and (Guess); one `TextView`. When the user taps the (+) or (-) Button increases or decreases the number displayed in the `TextView`, which starts at one and bound within one to nine. Tapping the guess Button, it displays a short message (a Toast) on the screen.

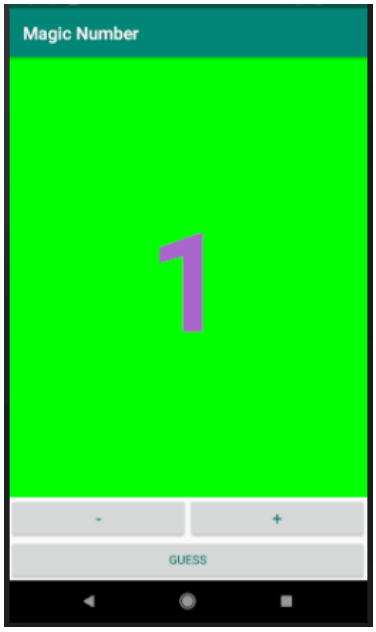
After each guess, we will output ONE hint (toast) according to the following conditions:

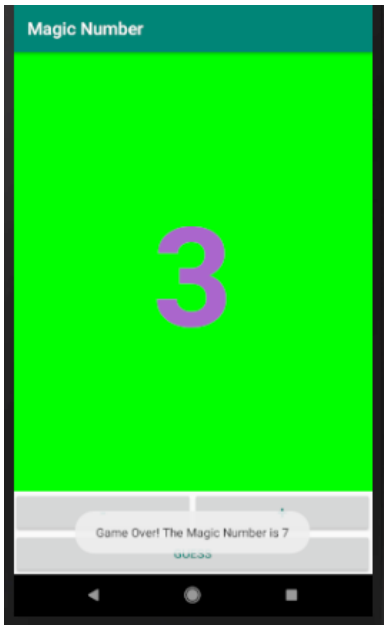
1) If the guessed number is larger than the magic number by at least 3, output "`Too large!`".
2) Otherwise, if the guessed number is smaller than the magic number by at least 3, output "`Too small!`".
3) Otherwise, if the difference between the guessed number and the magic number is within -2 to -1 or 1 to 2, output "`Close!`"
4) Finally, if the guessed number is equal to the magic number, output "`Bingo! You made it in X guess(es)!`" where X is the number of guesses the user made.
5) The program should end if condition (4) above is met.
6) However, if the user ultimately fails to guess the magic number correctly in 3 guesses, output "`Game Over! The Magic Number is Y`" where Y is the magic number randomly generated.

Disable all the button clicks after a correct guess or game over.

Please randomly set the magic number as an integer ranging from 1 to 9 in this lab.

Here's what the finished app looks like:





Magic Number App after tapping the Guess button is shown on the right.

1. Create and explore a new project

In this lab, you design and implement a project for the MagicNumber app.

1.1 Create the Android Studio project

1) Start **Android Studio** and create a new project.

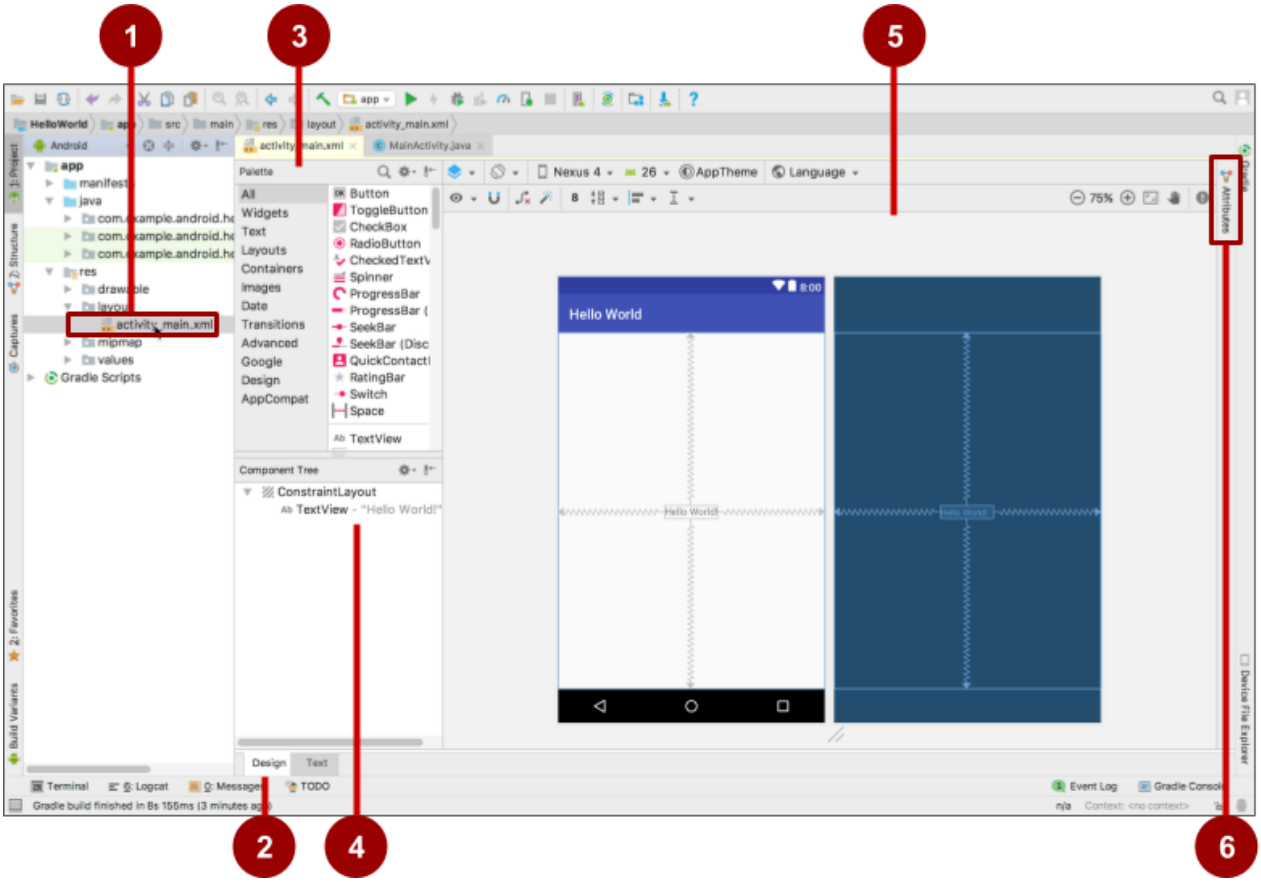
Attribute	**Value**
Application Name	**Magic Number**
Company/Domain Name	**edu.cuhk.csci3310**
Phone and Tablet Minimum SDK	**API29: Android 10.0(Q)**
Template	**Empty Activity**

2) Select **Run > Run app**
or click the **Run icon** in the toolbar to build and execute the app on the emulator or your device.

1.2 Explore the layout editor

Android Studio provides the layout editor for you to drag user interface (UI) elements to a visual design and blueprint view, position them in the layout, add constraints, and set attributes.

Explore the layout editor, and refer to the figure below as you follow the numbered steps:



1. In the **Project > Android** pane, within the **app > res > layout** folder, you can locate and open the **activity_main.xml** file for coding.
2. The **Design** tab is used to manipulate elements and the layout, and the **Text** tab is used to edit the XML code for the layout.
3. The **Palettes** pane shows UI elements that you can use in your app's layout.
4. The **Component tree** pane shows the tree hierarchy of UI elements - View elements, in which a child inherits the attributes of its parent.
5. The **Design and Blueprint** panes of the layout editor shows the UI elements in the layout.
6. The **Attributes** tab displays the **Attributes** pane for setting properties for a UI element.

2: Add View elements in the layout editor

In this task, you create the UI layout for the MagicNumber app in the layout editor using the `ConstraintLayout` features. *Constraints* determine the position of a UI element within the layout.

A constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline. You create the constraints semi-automatically or manually.

2.1 Examine the element constraints

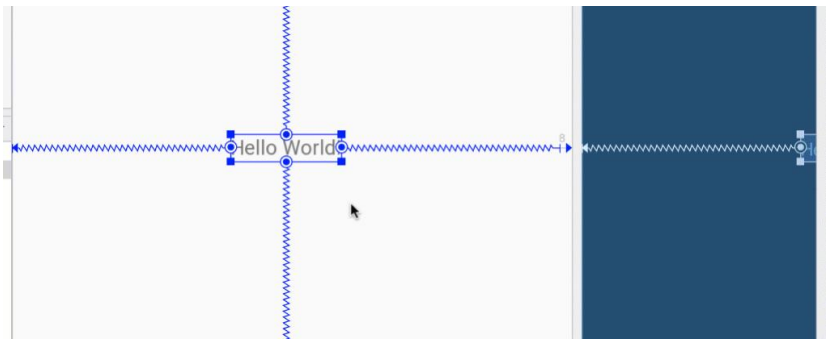
1) Open `activity_main.xml` from the **Project > Android**

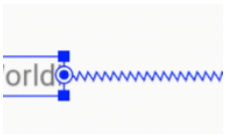
Ensure the **Autoconnect** tool is not disabled.

2) Select `TextView` in the Component Tree pane to highlight the "**Hello World**" `TextView` in the design and blueprint panes.

3) Click the circular handle on the right side of the `TextView` to delete the horizontal constraint,

that constraint binds the view to the right side of the layout.





2.2 Add a Button to the layout

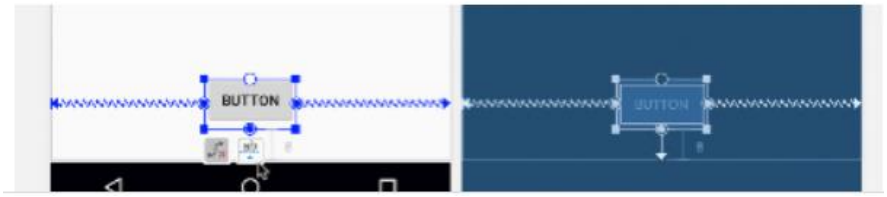
Follow these steps to add a `Button`:

1) Remove the default `TextView` to get a blank layout

2) Drag another `Button` from the **Palette** pane to the middle of the layout.

Autoconnect may provide the horizontal constraints for you (if not, drag them yourself).

3) Drag a vertical constraint to the bottom of the layout.



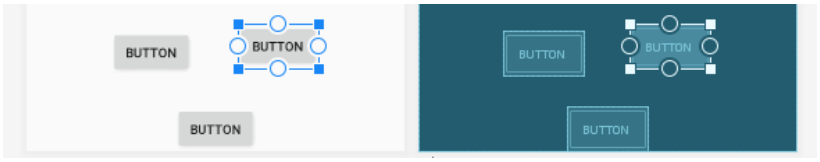
3: Create a chain of Button views

In this task, you add two more `Button` views. The two `Button` views are horizontally aligned with each other, thus be in a *chain*, and vertically constrained with the first `Button` view below.

3.1 Add a second and third Button to the layout

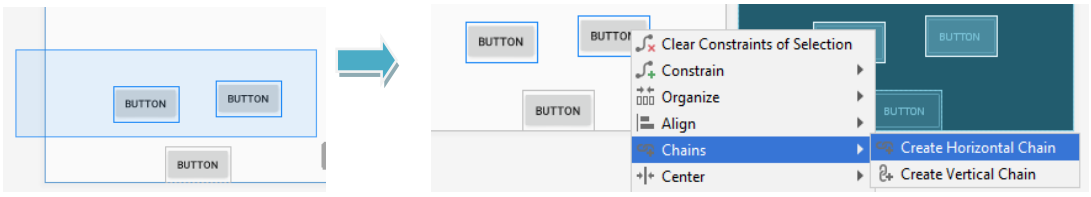
1) Drag a `Button` from the **Palette** pane to any position in the layout.
If you drop the `Button` in the top middle area of the layout, constraints may automatically appear.

2) Drag another `Button` from the **Palette** pane to any position in the layout.

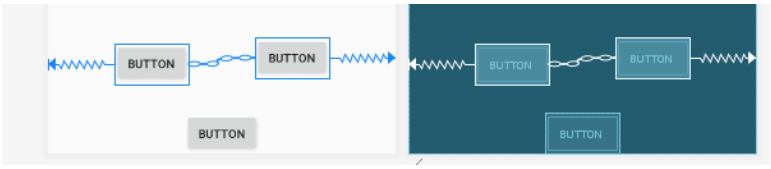


3.2 Create a chain and constrain it to the height of Box Two

1) Select all two new `Button` views, right-click, and select **Chains** > **Create Horizontal Chain**.

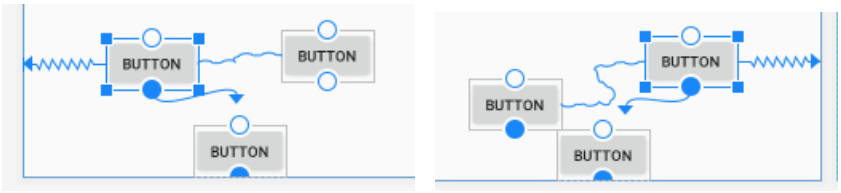


This creates a horizontal chain that extends from both `Button` views to the layout.

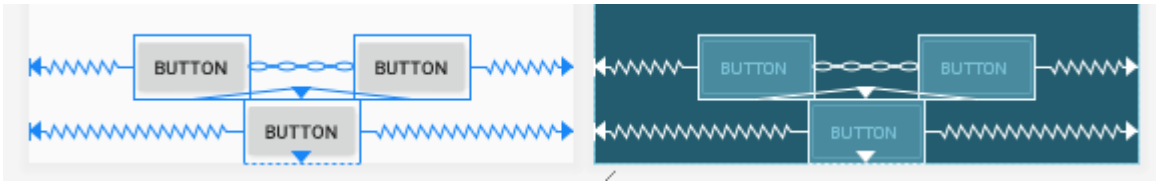


2) Add a constraint that extends from both `Button` views the top of the bottommost `Button`.

This removes the existing top constraint and replaces it with the new constraint. You don't have to delete the constraint explicitly.



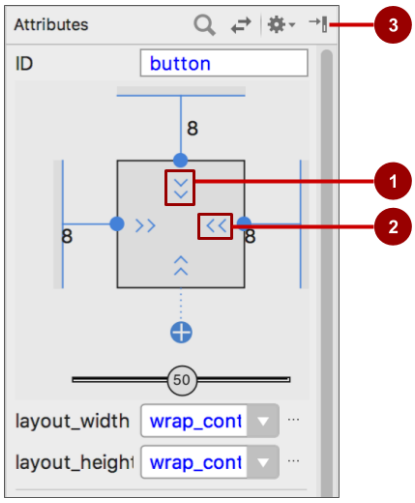
Observe that the two-button views are now constrained to the top of the **Button** below.



4: Change UI element attributes

In this task, you enter new values and change values for important `Button` attributes, which are applicable to most `View` types.

In the layout editor, one can drag the handles on each of the four corners of a `View` to resize it, but such hardcoded dimensions (width/height) can't adapt to different content and screen sizes so you should avoid.



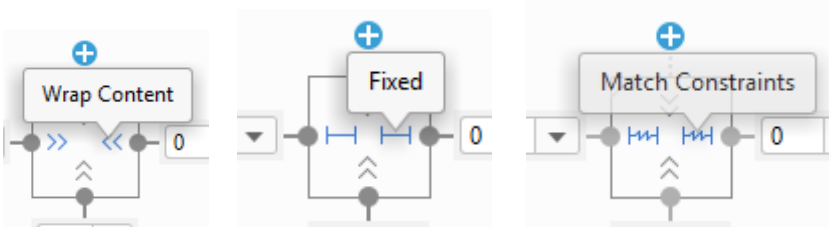
Instead, use the **Attributes** pane on the right side of the layout editor to select a sizing mode. The **Attributes** pane includes a square sizing panel called the *view inspector* at the top. The symbols inside the square represent the height and width settings of the Sizing panel in the Attributes pane:

1. **Height control**. This control specifies the layout_height attribute and appears in two segments on the top and bottom sides of the square. The angles indicate that this control is set to wrap_content, which means the `View` will expand vertically as needed to fit its contents. The "8" indicates a standard margin set to 8dp.
2. **Width control**. This control specifies the layout_width. As indicated by angles that this control is also set to wrap_content, which means the View will expand horizontally as needed to fit its contents, up to a margin of 8dp.

3. **Attributes** pane close button. Click to close the pane.

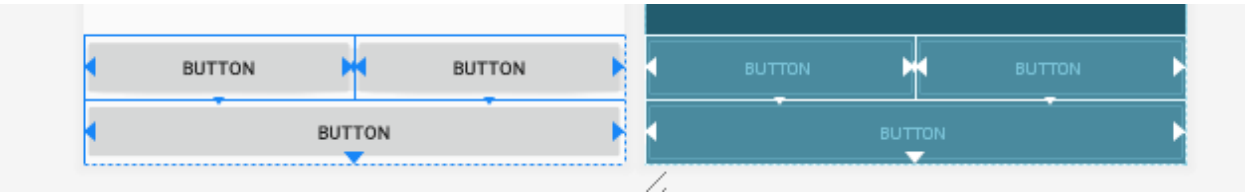
4.1 Change the Button size

1) Select the top `Button` in the **Component Tree** pane.
2) Click the **Attributes** tab on the right side of the layout editor window.
3) Click the width control twice—the first click changes it to **Fixed** with straight lines, and the second click changes it to **Match Constraints** with spring coils.



4) Select the second and third `Button` views, and make the same changes to the **layout_width** as in the previous step,

as shown in the figure below.

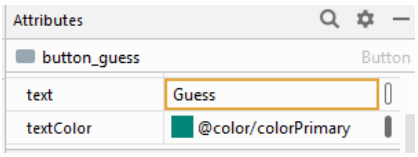


4.2 Change the Button attributes

To identify each `View` uniquely within an Activity layout, each `View` or `View` subclass (such as `Button`) needs a unique ID. And to be of any use, the `Button` elements need text. `View` elements can also have backgrounds that can be colors or images.

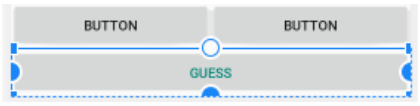
The **Attributes** pane offers access to all of the attributes you can assign to a `View` element:

1) After selecting the first `Button`, edit the ID field at the top of the **Attributes** pane to **button_guess**,



the `android:id` attribute, which is used to identify the element in the layout.

2) Edit the `text` attribute to **Guess**.



3) Set the `textColor` attribute to **@color/colorPrimary**.

(As you enter **@c**, choices appear for easy selection.)

4) Perform the same attribute changes for the second `Button`, using **button_minus** as the ID, for the text attribute, the `textSize` attribute to **20sp**

and the same colors for the background and text as the previous steps.

5) For the third Button, using **button_plus** as the ID, **+** for the text attribute, and the same text size and colors for the background and text as the previous steps.

5: Add a TextView and set its attributes

In this task, you add a `TextView` in the middle of the layout, constrain it horizontally and vertically and also change the attributes for the `TextView` in the **Attributes** pane. ConstraintLayout is to constrain elements relative to other elements.

5.1 Add a TextView and constraints

1) Drag a `TextView` from the **Palette** pane to the upper part of the layout, and

2) Drag a constraint from the top of the `TextView` to the top of the layout.

This constrains the `TextView` to be underneath the top margin.

3) Drag a constraint from the bottom of the `TextView` to the handle on the top of either the **(+)** or **(-)** `Button`, and from the sides of the `TextView` to the sides of the layout.

5.2 Set the TextView attributes

With the `TextView` selected, open the **Attributes** pane, if it is not already open. Set attributes for the `TextView` as follows:

1) Set the ID to **show_number**.

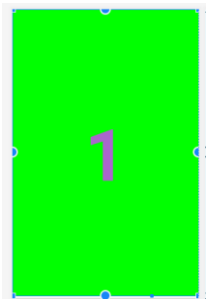
2) Set the text to **1**.

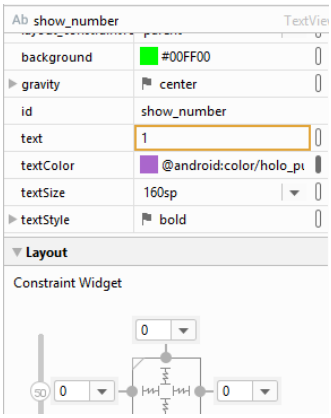
3) Set the textSize to **160sp**.

4) Set the textStyle to **B** (bold).

5) Change the horizontal and vertical view size controls to **match_constraint**.

(layout_width and layout_height)





6) Set the textColor to @android:color/holo_purple.

7) Scroll down the pane and click **View all attributes**, scroll down the second page of attributes to the background, and then enter **#00FF00** for a shade of green.

8) Scroll down to gravity, expand gravity, and select **center** (for enabling both center_vertical and center_horizontal).

6: Add onClick handlers for the buttons

In this task, you add a method for each `Button` in `MainActivity` that executes when the user taps the Button.

6.1 Add the onClick attribute and handler to each Button

In Android Studio you can specify the name of the method in the `onClick` field in the **Design** tab's **Attributes** pane.

You can also specify the name of the handler method in the XML editor by adding the `android:onClick` property to the `Button`.

You will use the latter method because you haven't yet created the handler methods, and the XML editor provides an automatic way to create those methods.

1) With the XML editor open (the **Text** tab), find the `Button` with the `android:id` set to **button_guess**.

```
<Button
        android:id="@+id/button_guess"
        android:layout_width="0dp"
        ...
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
```

1) Add the `android:onClick` attribute as **"showToast"** to the end of the `button_guess` element after the last attribute and before the /> end indicator:

```
android:onClick="showToast"/>
```

2) Click the red bulb icon that appears next to attribute. Select **Create click handler**, choose **MainActivity**, and click **OK**.

If the red bulb icon doesn't appear, click the method name ("showToast"). Press **Alt-Enter** (**Option-Enter** on the Mac), select **Create 'showToast(view)' in MainActivity**, and click **OK**.

This action creates a placeholder method stub for the `showToast()` method in `MainActivity`, as shown at the end of these steps.

3) Repeat the last two steps with the **button_plus** `Button`: Add the `android:onClick` attribute to the end, and add the click handler:

```
android:onClick="countUp" />
```

4) Similarly for the **button_minus** `Button`: Add the `android:onClick` attribute to the end, and add the click handler:

```
android:onClick="countDown" />
```

The XML code for the UI elements within the ConstraintLayout now looks like this:

```
    <Button
        android:id="@+id/button_guess"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Guess"
        android:textColor="@color/colorPrimary "
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:onClick="showToast"/>

    <Button
        android:id="@+id/button_plus"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="+"
        android:textColor="@color/colorPrimary "
        android:textSize="20"
        app:layout_constraintBottom_toTopOf="@+id/button_guess"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/button_minus"
        android:onClick="countUp" />

    <Button
        android:id="@+id/button_minus"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="-"
        android:textColor="@color/colorPrimary "
        android:textSize="20"
        app:layout_constraintBottom_toTopOf="@+id/button_guess"
        app:layout_constraintEnd_toStartOf="@+id/button_plus"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        android:onClick="countDown" />
```

If `MainActivity.java` is not already open, expand **java** in the Project > Android view, expand **edu.cuhk.csci3310.magicnumber**, and then double-click **MainActivity**. The code editor appears with the code in `MainActivity`:

```
package edu.cuhk.csci3310.magicnumber;
//import statement for different versions of Android Studio may differ
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {
```

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

    public void showToast(View view) {
    }

    public void countUp(View view) {
    }
    public void countDown(View view) {
    }
}
```

6.2 Edit the + Button handler and - Button handler

You will now edit the `countUp()` method—the **(+)** `Button` click handler in `MainActivity`—so that it displays the current count after **(+)** is tapped. Each tap increases the count by one.

The code for the handler must:

- Keep track of the count as it changes.
- Send the updated count to the `TextView` to display it.

Follow these steps to edit the **(+)** `Button` click handler:

1) Locate and add a `mGuessNumber++` within the newly created `countUp()` method.

This is to keep track of the count by a private member variable `mGuessNumber`. Each tap of the **(+)** button increases the value of this variable.

```
public void countUp(View view) {
    mGuessNumber++;
}
```

2) Click the red bulb icon (select the `mGuessNumber++` expression to show) and choose to **Create field 'mGuessNumber'** from the popup menu.

This creates a private member variable at the top of `MainActivity`, and Android Studio assumes that you want it to be an integer (int).

```
public class MainActivity extends AppCompatActivity {
    private int mGuessNumber;
```

3) Change the private member variable statement to initialize the variable to one.

```
public class MainActivity extends AppCompatActivity {
    private int mGuessNumber = 1;
```

4) Add a private member variable `mShowNumber`, for the reference of the `show_number` `TextView`,

which you will add to the click handler, collectively looks like:

```
public class MainActivity extends AppCompatActivity {
    private int mGuessNumber = 1;
    private TextView mShowNumber;
```

5) You can get a reference to the `TextView` using the ID you set in the layout file and assign it to mShowNumber.

In order to get the `TextView` reference **only once**, specify it in the `onCreate()` method. The `onCreate()` method is used to *inflate the layout*, which means to set the content view of the screen to the XML layout. You can also use it to get references to other UI elements in the layout, such as the `TextView`. Locate the onCreate() method in `MainActivity`.

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

6) Add a `findViewById` statement to the end of the method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mShowNumber = (TextView) findViewById(R.id.show_number);
}
```

A `View`, like a string, is a resource that can have an id. The `findViewById` call takes the ID of a view as its parameter and returns the `View`. Because the method returns a `View`, you have to cast the result to the view type you expect, in this case (`TextView`).

Now that you have assigned to mShowNumber the `TextView`, you can use the variable to set the text in the `TextView` to the value of the mGuessNumber variable.

7) Add a line to set up the text of the `TextView` within the `countUp()` method.

```
if (mShowNumber != null)
            mShowNumber.setText(Integer.toString(mGuessNumber));
```

8) Run the app to verify that the number increases when you tap the **(+)** button.

9) Repeat the steps for the handler in tapping the **(-)** button.

The Magic Number app showing the number as you tap the **(+)** or **(-)** button.

6.3 Edit the Guess Button handler

You will now edit the showToast() method—the **Guess** Button click handler in `MainActivity`—so that it shows a message.

A Toast is a small message-showing popup window, which fills only the amount of space required for the message.

Follow these steps to edit the **Guess** `Button` click handler:

1) Locate the newly created `showToast()` method.

```
public void showToast(View view) {
}
```

2) To create an instance of a `Toast`, call the `makeText()` factory method on the `Toast` class.

```
public void showToast(View view) {
    Toast toast = Toast.makeText(
}
```

The above statement is incomplete until you finish all of the steps below.

Supply the context of the app `Activity` use this as already within its context.

```
Toast toast = Toast.makeText(this,
```

Supply the message to display, such as a `String` named `toast_message`.

```
Toast toast = Toast.makeText(this, toast_message,
```

Supply a duration for the display. E.g., `Toast.LENGTH_SHORT` displays the toast for a relatively short time, ~2s.

```
Toast toast = Toast.makeText(this, toast_message, Toast.LENGTH_SHORT);
```

3) Show the Toast by calling `show()`. The following is the entire showToast() method.

```
public void showToast(View view) {
   Toast toast = Toast.makeText(this, toast_message, Toast.LENGTH_SHORT);
   toast.show();
}
```

Run the app and verify that the Toast message appears when the **Toast** button is tapped.

4) Update the toast message according to the number guessing conditions
[Hints: stated in the TODO section.]

5) Disable all button click if the user ultimately fails to guess the magic number correctly in 3 guesses.

The Magic Number app showing a Toast message.

Congrats! You have created the Magic Number app!