



Maydm

The Quizzler: Add Icons

Add Icons for Higher Interaction

We can add images to our apps to create highly impressive apps.

Different Android phones have different levels of pixel density. (That refers to how many pixels are squeezed into the device screen). The higher the pixel density the higher resolution images we can serve to our user.

Android will figure out density resolution for us, all we need to do is provide the app the same images with different resolutions.

Adding Directories

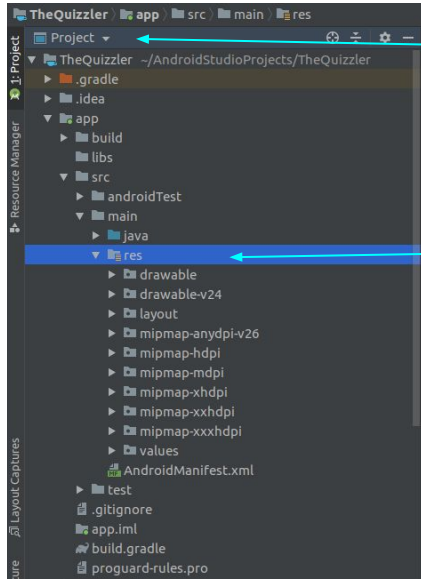
You'll find four folders containing left and right arrows in .png images. These images will be found in a GitHub Repository at <https://github.com/Maydm/appetite-for-apps/tree/master/TheQuizzler/app/src/main/res>

The folders are `drawable-hdpi`, `drawable-mdpi`, `drawable-xhdpi`, and `drawable-xxhdpi`. These folders hold the same images, but they are of different resolutions for different resolution screens.

We are going to add four folders into the Android directory
``TheQuizzler/app/src/main/res``

Copy Folders

There are a couple of ways we can move the directories into the Android directory. We can download the directories onto our local hard drive and then copy the directories and paste the folders into the proper directory or drag and drop.



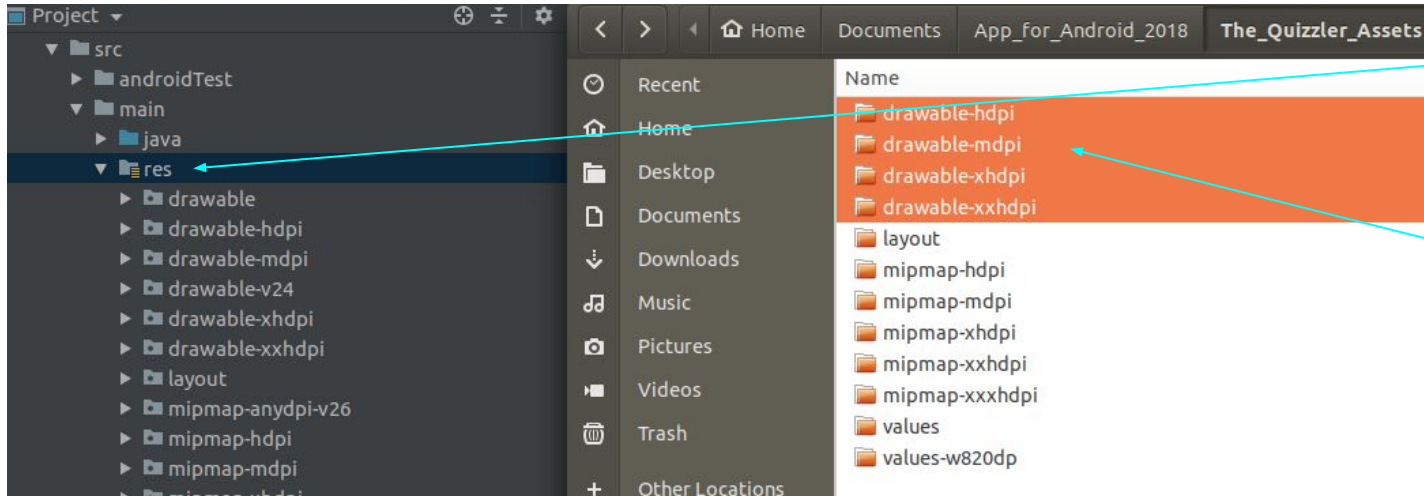
Be sure you're in the Project view

This is the ~~Android~~ directory you're looking for
TheQuizzler/app/src/main/res

Copy Folders

There are a couple of ways we can move the directories into the Android directory. Select the four folders, right-click Copy and then hover over "res" in Android Studio and right-click Paste.

OR select the four folders and drag and drop them into the "res" folder.

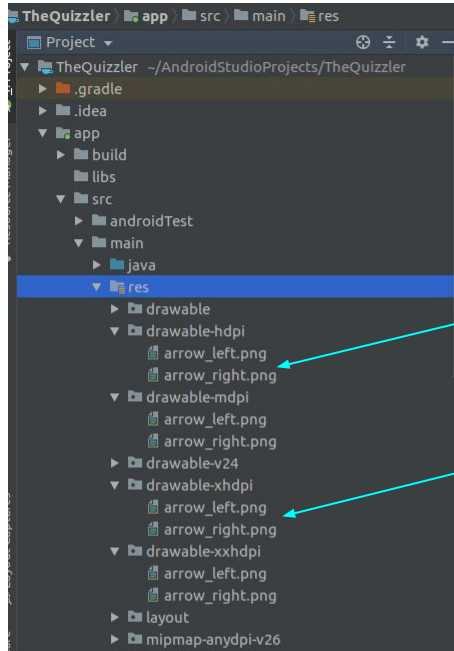


Right-click and paste or drag and drop here

Select these 4 folders and right-click copy or drag and drop to 'res' with mouse

The images are now in your Android Project

You can now see the left and right arrow .png images inside the directories in the 'res' directory.



If your directory looks similar to this then great work! The images are ready to be used in your app!

A bit about App Images

Images can be in .jpg, .png, or .gif formats.

Images must be all lowercase and have no spaces.

When images are added to the /res/drawable directory, Android automatically assigns resource IDs for the images. You don't need to worry about the user's screen resolution. Android will figure that out for you.

Reference Image Resources in XML

Just as we reference a string resource with @string, we reference drawable resources with @drawable/

Let's reference the right arrow with the @drawable resource ID in activity_main.xml.

Find the next button widget and we'll add two lines so Android will draw the right arrow.

<Button

android:id="@+id/next_button"

android:text="@string/next_button"

android:drawableRight="@drawable/arrow_right"

android:drawablePadding="4dp"

app:layout_constraintTop_toBottomOf="@+id/question_text_view" />

Add these two lines to the existing XML for the next_button widget

Emulate the App to see your Arrow in Action

Emulate the app on your computer or device to see your new code in action.

Touch through a few questions and then turn the device. Can you identify a bug?

The app reverts back to the first question. We can fix this using an activity lifecycle.

But first, let's work on the UI to create a more aesthetically pleasing app.

Troubleshooting:

If Android tells you the emulator is running, but it isn't then use the terminal to delete the .lock files with the command:

```
$ cd ..
```

```
/home$ USER/.android/avd/PHONE_TYPE.avd/*.lock
```

Add a Previous Button Widget

This app is really coming together. Is there anything else we could add to create a stronger user experience?

Perhaps a Previous Button?

Let's add that. We can start by creating a new Button widget in activity_main.xml

You can use the Next Button widget code as a template for what this code will look like.

Try working on this yourself for a moment and then we'll show what the code might look like in the next slide.

Previous Button Widget XML Code

<Button

```
    android:id="@+id/prev_button"  
    android:layout_width="90dp"  
    android:layout_height="48dp"  
    android:drawableLeft="@drawable/arrow_left"  
    android:text="Prev" />
```

Android will automatically add margins and constraints when you use the Preview to move the button into place.

Now that the widget is in place, let's go to MainActivity.java and add a new PrevButton Class.

Previous Button Class

To create the code for the PrevButton class look at the NextButton as a template. There will be one major difference between NextButton and PrevButton. Do you know what it is?

Take a few moments to see what you can create for code in MainActivity.java

If you think you have a solution test it in the emulator.

Previous Button Class

How is your code looking? Notice any interesting bugs in the emulator?

Here is a side-by-side comparison of the NextButton & PrevButton Classes

```
mNextButton = (Button)
findViewById(R.id.next_button);
mNextButton.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex + 1);
        updateQuestion();
    }
});
```

```
mPrevButton = (Button)
findViewById(R.id.prev_button);
mPrevButton.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex - 1);
        updateQuestion();
    }
});
```

Bug Report

Run the newest version of our app in the emulator. Do you notice any new bugs?
What happens when we use Previous on the first question?

The app crashes! Android doesn't like going to the negative index. So how can we fix that?

We have an array of questions in the question bank. As far as the index is concerned, the first question ID is 0 as in ZERO. The second question has an ID of 1, and so on and so forth. There is no index ID of -1. That would mean there is nothing in the array and that's why we receive the error. So, what can we add in the PrevButton class to disable the Previous Button if we're at index 0 in the question bank?

Time for a Conditional

If you suggested that we add a conditional (or an “if”) statement then you’re absolutely correct!

Let’s tell the PrevButton class to only decrease the currentIndex if the index is not at zero.

Can you code that? Give it a try and we’ll show the solution on the next slide.

PrevButton Class with Condition

```
mPrevButton = (Button) findViewById(R.id.prev_button);  
mPrevButton.setOnClickListener(new View.OnClickListener() {
```

```
@Override
```

```
public void onClick(View v) {
```

```
    if (mCurrentIndex > 0) {
```

```
        mCurrentIndex = (mCurrentIndex - 1);
```

```
        updateQuestion();
```

```
    }
```

```
}
```

```
});
```

Condition such that
Android only
updates the
currentIndex and
Question if the index
is greater than zero.

Curly brace to close the if statement

Additional UX Updates

Run the emulator again and test our conditional. Now the previous button should only work if we're not already looking at the first question. There are other ways we could resolve this bug. For example, we could store the length of the question bank array and set a conditional statement that would instruct the code to cycle to the highest index if it was at index 0. But this is good for now.

What else can we update?

Additional UX Updates

We could make the UI respond to a touch of the question, itself. So if a user taps the question the app will increase the `currentIndex` of the question bank and update the question with the next question in the array.

How might we do this?

Refactoring UI

One way to accomplish this next UI Update will be to consider the code for NextButton (once again) and the code for the `question_text_view`.

Currently `QuestionTextView` is a single line of code which takes the ID of the question and displays it in the `TextView` Widget, which we created in `activity_main.xml`.

```
mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
```

What if we provided `mQuestionTextView` an event listener? If that event listener looked like the event listener in Next Button what would be the end result?

Let's try this for a few moments and then we'll compare our results with the sample code in the next slide.

New Code for QuestionTextView

Our newest event listener may look something like this. You may notice that we're violating our DRY principle. Do you see the repetition of code? How might we refactor the code to avoid this?

```
mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
mQuestionTextView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
        updateQuestion();
    }
});
```

DRY Violation

```
activity_main.xml x strings.xml x MainActivity.java x Question.java x
view
28
29 @Override
30 protected void onCreate(Bundle savedInstanceState) {
31     super.onCreate(savedInstanceState);
32     setContentView(R.layout.activity_main);
33
34     mQuestionText = (TextView) findViewById(R.id.question_text);
35     mQuestionText.setOnClickListener(new View.OnClickListener() {
36
37         @Override
38         public void onClick(View v) {
39             mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
40             updateQuestion();
41         }
42     });
43
44     mTrueButton = (Button) findViewById(R.id.true_button);
45     mTrueButton.setOnClickListener(new View.OnClickListener() {
46
47         @Override
48         public void onClick(View v) {
49             checkAnswer( userPressedTrue: true);
50         }
51     });
52
53     mFalseButton = (Button) findViewById(R.id.false_button);
54     mFalseButton.setOnClickListener(new View.OnClickListener() {
55
56         @Override
57         public void onClick(View v) {
58             checkAnswer( userPressedTrue: false);
59         }
60     });
61
62     mNextButton = (Button) findViewById(R.id.next_button);
63     mNextButton.setOnClickListener(new View.OnClickListener() {
64
65         @Override
66         public void onClick(View v) {
67             mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
68             updateQuestion();
69         }
70     });
71
72     updateQuestion();
73 }
```

Remember, similar code in two different places indicates that we can create a new function and then call the function

Creating a new function to avoid DRY

When we create a new function we want to give it a name that helps another human quickly understand what the function does.

Also, functions should do one thing and one thing only to avoid hard to find bugs in the future. As it happens, this function will contain all of one line. Add this new function after the `updateQuestion()` function.

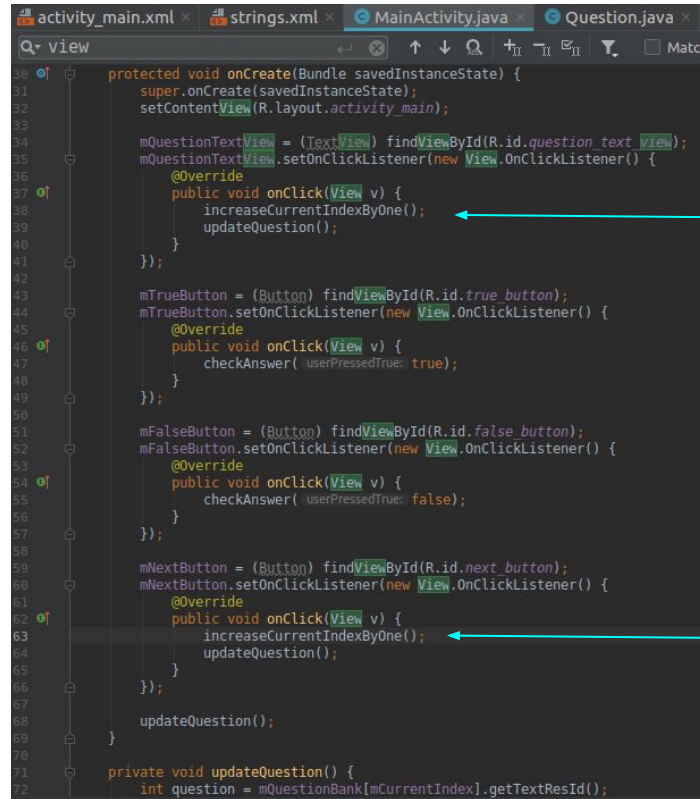
```
private void increaseCurrentIndexByOne() {  
    mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;  
}
```

We won't include the `updateQuestion()` method in our new function, although we could. By not including the `updateQuestion()` we create more modular code.

Update repeating code

Update `mQuestionTextView` and `mNextButton` with the `increaseCurrentIndexByOne()` method.

We could create another function that called `increaseCurrentIndexByOne()` followed by `updateQuestion()` and then call that new function something like `increaseIndexAndUpdateQuestion()` but we won't do that here. If you'd like to make that change go for it! However, we won't show what that would look like on the slides.

A screenshot of an IDE showing the MainActivity.java file. The code is in Java and implements an Android activity. Two red arrows point to specific lines of code. The first arrow points to line 38, which is the `increaseCurrentIndexByOne()` call inside the `onClick` method for `mQuestionTextView`. The second arrow points to line 63, which is the `increaseCurrentIndexByOne()` call inside the `onClick` method for `mNextButton`. The code also shows the initialization of `mQuestionTextView`, `mTrueButton`, `mFalseButton`, and `mNextButton`, and the `updateQuestion()` method.

```
10 protected void onCreate(Bundle savedInstanceState) {
11     super.onCreate(savedInstanceState);
12     setContentView(R.layout.activity_main);
13
14     mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
15     mQuestionTextView.setOnClickListener(new View.OnClickListener() {
16         @Override
17         public void onClick(View v) {
18             increaseCurrentIndexByOne();
19             updateQuestion();
20         }
21     });
22
23     mTrueButton = (Button) findViewById(R.id.true_button);
24     mTrueButton.setOnClickListener(new View.OnClickListener() {
25         @Override
26         public void onClick(View v) {
27             checkAnswer( userPressedTrue: true);
28         }
29     });
30
31     mFalseButton = (Button) findViewById(R.id.false_button);
32     mFalseButton.setOnClickListener(new View.OnClickListener() {
33         @Override
34         public void onClick(View v) {
35             checkAnswer( userPressedTrue: false);
36         }
37     });
38
39     mNextButton = (Button) findViewById(R.id.next_button);
40     mNextButton.setOnClickListener(new View.OnClickListener() {
41         @Override
42         public void onClick(View v) {
43             increaseCurrentIndexByOne();
44             updateQuestion();
45         }
46     });
47
48     updateQuestion();
49 }
50
51 private void updateQuestion() {
52     int question = mQuestionBank[mCurrentIndex].getTextResId();
53 }
```

Convert Buttons to ImageButtons

With the rise of emojis and symbol usage, many users instantly recognize the meaning of symbols without the help of any language. Best of all, symbols are often cross-cultural, meaning people of many different cultures and languages can understand the intention of symbols. For example, a smiley emoji means the same thing no matter what part of the planet you are from.

With that in mind, let's refactor the Previous and Next buttons into image buttons that only show the left and right arrows.

Convert Buttons to ImageButtons

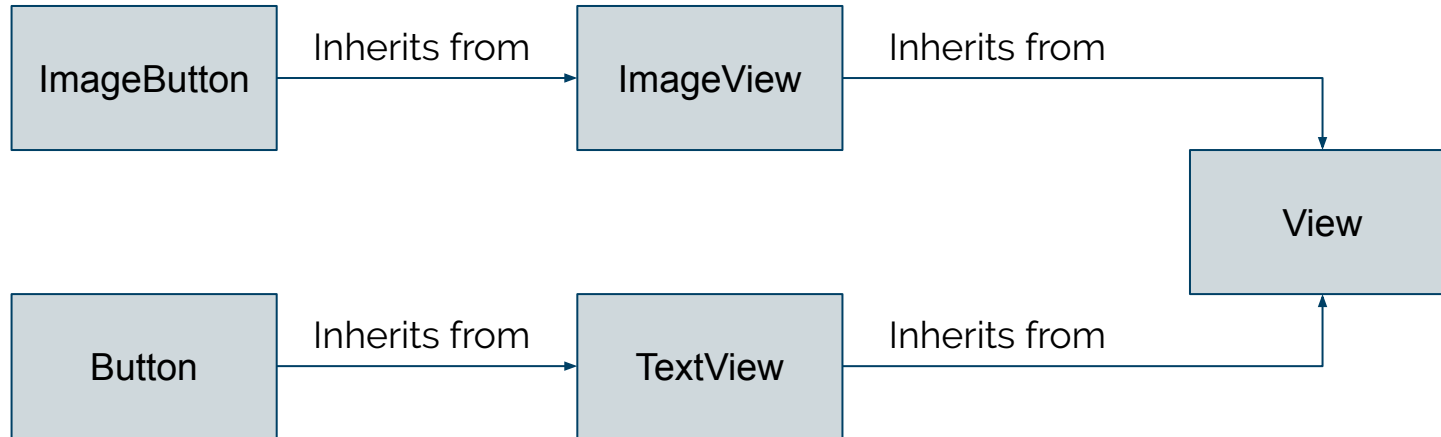
A few steps are required to convert Buttons to ImageButton widgets.

First, in `activity_main.xml` we'll change the name of "Button" to "ImageButton"

ImageButton is a completely different widget as far as Android is concerned, and so this will require a new import. Can you import the widget using only the prompts from Android Studio?

We're also going to need to import something called `ImageView`. It may not seem apparent at first, but for this second import is because `ImageButton` and `Button` inherit properties from different widgets. We'll draw a diagram on the next slide to demonstrate.

View Widget Children Diagram



All of these widgets inherit properties from the View, but Button and ImageButton inherit from different widgets. For this reason, when updating Buttons to ImageButtons we must also update the existing TextView attributes to an ImageView attribute

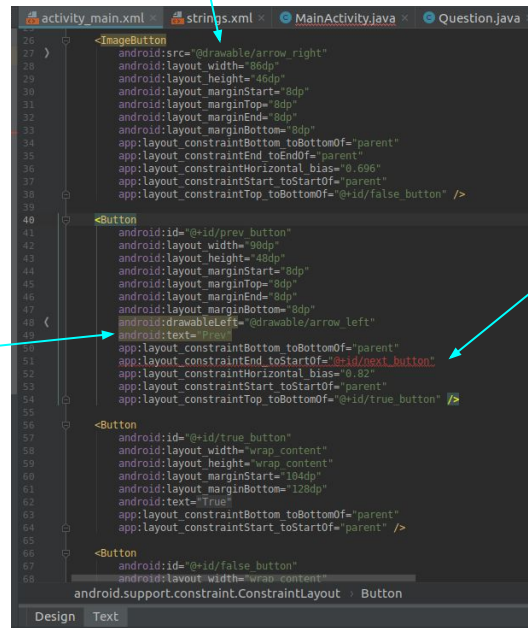
Updated XML for the ImageButton Widget

This screenshot shows the differences in the ImageButton and Button Widgets.

The first line in the ImageButton is an ImageView attribute. We'll remove the TextView attributes of android:text, android:drawableRight/Left and android:drawablePadding

The new ImageView attribute

We'll remove the highlighted attributes



Some of your constraints may be lost in this process. We can safely remove the layout_constraint attributes. When we move the widgets in the preview they will reset.

Convert Buttons to ImageButtons

This is a sample of the code for the Left ImageButton. (layout attributes have been removed for brevity)

```
<ImageButton
```

```
    android:id="@+id/prev_button"
```

```
    android:src="@drawable/arrow_left"
```

```
    android:drawableLeft="@drawable/arrow_left"
```

```
    android:drawablePadding="4dp"
```

```
    android:text="Prev" />
```

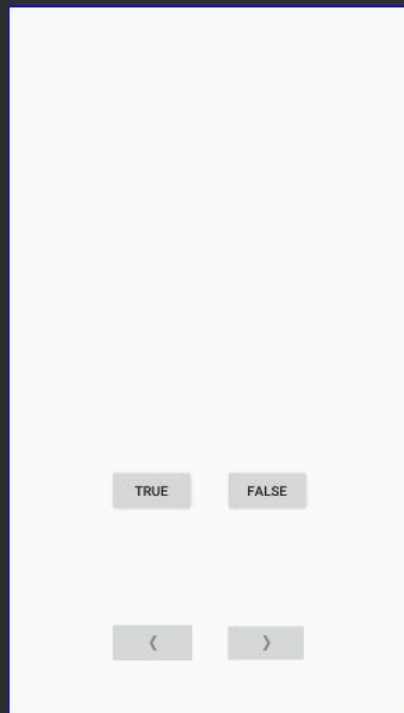
Remove the TextView
attributes.

Final ImageButton XML

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.483" />

<ImageButton
    android:id="@+id/next_button"
    android:layout_width="86dp"
    android:layout_height="46dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:src="@drawable/arrow_right"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.695"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/false_button"
    app:layout_constraintVertical_bias="0.703" />

<ImageButton
    android:id="@+id/prev_button"
    android:layout_width="90dp"
    android:layout_height="48dp"
    android:layout_marginStart="104dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:src="@drawable/arrow_left"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/next_button"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/true_button"
    app:layout_constraintVertical_bias="0.706" />
```



Wire MainActivity.java with ImageButtons

Now that the XML is complete, we need to update MainActivity so it looks for ImageButtons for Next and Previous rather than the former Button widgets.

If you look at the following snippet can you determine what needs to be updated and with what?

```
private Button mTrueButton;  
private Button mFalseButton;  
private Button mNextButton;  
private Button mPrevButton;  
private TextView mQuestionTextView;
```

Wire MainActivity.java with ImageButtons

If you think we need to update the Button Widget variables for mPrevButton and mNextButton with “ImageButton” then you are absolutely correct.

```
private Button mTrueButton;  
private Button mFalseButton;  
private ImageButton mNextButton;  
private ImageButton mPrevButton;  
private TextView mQuestionTextView;
```



Android will try to help you fix bugs in the code with red light bulbs and error messages

We also need to update where we store the ID in the Button variables. A quick way to find these is by using `ctrl + f` to find every instance of the word ‘Button’. This will help you find the buttons that need to be updated to ImageButtons.

```
mNextButton = (ImageButton) findViewById(R.id.next_button);  
mPrevButton = (ImageButton) findViewById(R.id.prev_button);
```

Accessibility

We might take it for granted, but all sorts of people use Apps. That includes the deaf, visually impaired, and other humans with accessibility issues. We want our apps to be friendly to all peoples, so let's be sure to add a `contentDescription` attribute in the `ImageButtons` so their phone can tell them what the buttons mean if necessary. Go to `activity_main.xml` and add a single line in each of the `ImageButton` widgets.

If we hard code the string such as: `android:contentDescription="Previous"`

Then Android will ask us to use a String resource instead. So, let's update `strings.xml` with a new string resource for "Previous" and use the string resources for our `contentDescription` in both `ImageButtons`.

`android:contentDescription="@string/prev_button"`

Emulate

Run the app on your device or the emulator and take a moment to test all of the buttons.

If everything works then great work! If something isn't acting right then we can run the debugger to work to locate any issues. You can also look for red light bulbs, indicating an issue that needs to be fixed.

This was a long lesson and you made a lot of updates and thought about how to add new features on your own. That's hard work! Give yourself a pat on the back and stretch out a little if you need to.