Mobile App Design Project
American Sign Language Alphabet App
(Resources Version)

Directions:

This App will take a phrase entered by the user and then display the Sign Language symbol for each letter in the phrase. We will use XML to design the User Interface. The User Interface will have an Edit Text Field, A Button, and an Image View Field. The Code for this App will consist of one Class Called Translate which will hold the Data and Functions for the User Interactions with the Screen.

User Interface Objects and instance names:

1. ImageView - aslImages
2. TextView - displayText
3. Button – translate
4. EditText - enterText

User Interactions:

1. Typing the Phrase into the Edit Text box
2. Touching (Clicking) the Translate Button
3. Touching(Clicking) the ImageView object to advance the Pictures

App Functions:

1. React to Typing and Place text in the EditText Box
2. Retrieve Text from the EditText Object
    a. Convert Text to Editable
    b. Convert Editable to String
    c. Convert all the characters in the String to lower Case
3. Translate the Letters in the Translate String – Display Images in ImageView in sequence based on User Touch Event

App Resources:

This App will use .gif images of ASL Sign Language letters. Apps and Applications will store resources (Data, Sound, Images, Movies, and Other Data Objects) in folders in a directory called 'res'. The 'res' stands for resources and in this project we will practice adding Data to the 'res' folder and then calling this data using 'R.drawable' command.

Class Main Extends Activity

Fields:

Android OS Objects:
Button translate: Calls the translate function when pressed
EditText enterText: Field where User enters phrase to translate
TextView displayText: Displays Text during run of Translation
ImageView aslImages: Displays the Images during run of Translation

Data for Translation Function
Int phraseIndex: Used to track location in Translation String
String letters: Message to be translated
String display: Holds the letters to be displayed in the displayText Object

Array char letterIndex: Stores in Hardcode the alphabet in lower case
Array int aslPics: Stores references to the R.drawable images for the ASL Alphabet.

Note that letterIndex and aslPics are parallel Arrays.

OnCreate Function (Acts similar to a Constructor in Android Activities)

super.onCreate
setContentView to main activity layout

Bind the Android Object instances to the layout objects
Translate -> R.id.buttonTranslate
enterText -> R.id.textInput
displayTest -> R.id.displayText
aslImages -> R.id.aslViewer

enterText.setSelectAllOnFocus(true) -> Selects all text in enterText

setString Function (Moves Text from enterText to letters string)

(Triggered by Touch Event on translate Button Object)
Hide Keyboard
Set phraseIndex to 0
Set display String to “”
Get the Input Text
Set Input Text to letters String

translateLetter Function(walks through letters string and displays images)

Check if EditText phrase was translated
Fetch the Current Letter according to phraseIndex
Display the ASL image corresponding to the Current Letter index
Add the current letter to the display String
Set the displayText to the display String

Check to see if you reach the end of the phrase and reset the phrase at end.

User Interface and XML Design:

The User Interface will have the following Objects: (Note that these correspond to the Android Main Class objects we defined in the fields. In Android Apps – we 'create' objects three times:

1. In the XML User Interface view
2. In the Fields we define within the Classes
3. In the Constructor or Functions within the Code (This is where we 'bind' them to the XML user interface).

Any errors in these three steps will cause the App to crash. (And sometimes you will not get a specific error message describing the nature of the crash). When the App does not run – double check that you have done the "Big Three" definitions. (XML, Fields, Functions/Binding).

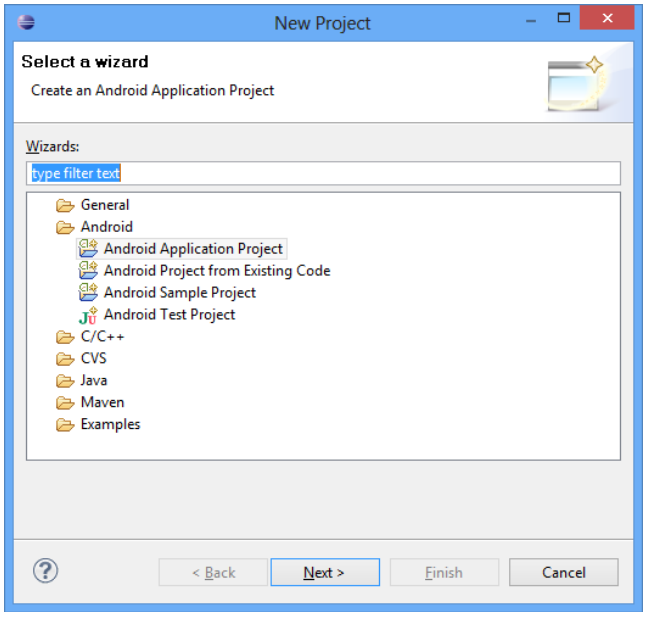
XML User Interface Objects in ASL Alphabet App:

ImageView named aslViewer
TextView named displayText
EditText named textInput
Button named buttonTranslate

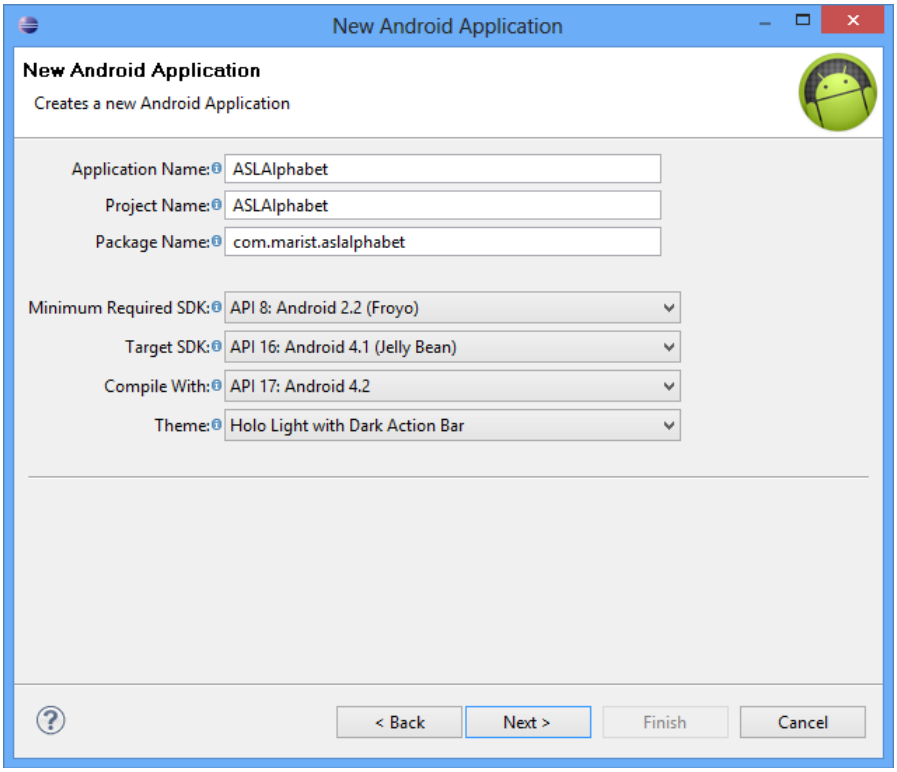
Directions:

Phase 1: Create the App Project

1. Start Eclipse and select New Project -> “Android Application Project”



2. Fill out the Wizard form as shown below:
    a. Application Name: ASLAlphabet
    b. Project Name: ASLAlphabet
    c. Package name: com.marist.aslalphabet

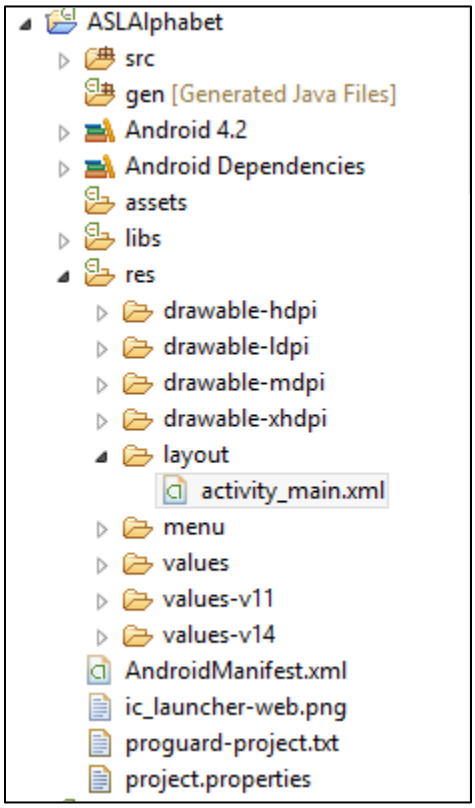


3. Click next at the Configure Project Window
4. Click next at the Configure Launcher Icon. (We can set this later)
5. Select “Blank Activity” on Create Activity and Click Next.
6. Name the Activity “Main” and Click Finish

Phase 2: Write the XML Code to Define the User Interface

We need to define the User Interface elements with the XML Code. For this project, we will type in the User Interface setup in XML. While Android and Eclipse do provide a GUI interface builder – using XML will give you more control in the layout and look of the User Interface.

1. Go to res->layout->activity_main.xml by left clicking on 'activity_main.xml."



2. Click the activity_main.xml tab in the Editor Window:

3. Delete the existing code and type the following lines to start creating a Table Layout: (Lines 1 to 5)

```
1  <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      tools:context=".Main" >
6
```

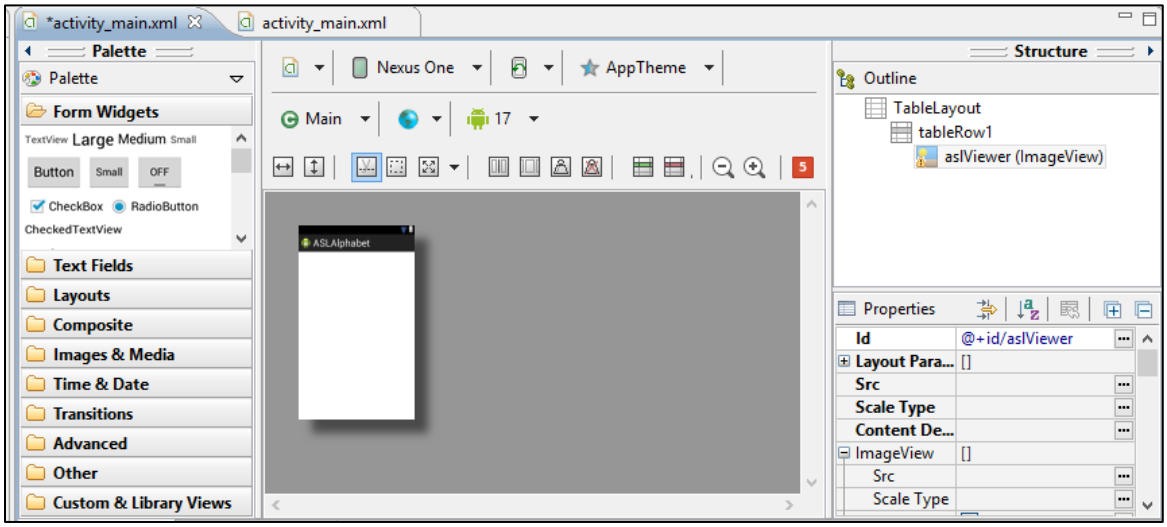
4. Type the following lines to begin defining the Table Row: (Lines 6 to 11)

```
6
7      <TableRow
8          android:id="@+id/tableRow1"
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content" >
11
```

5. Type the following Lines to define the ImageViewer and Complete the Table Row and Table Layout. (Lines 11 to 22)

```
11
12         <ImageView
13             android:id="@+id/aslViewer"
14             android:layout_width="wrap_content"
15             android:layout_height="300dp"
16             android:layout_span = "2"
17             android:onClick = "translateLetter"
18             />
19
20     </TableRow>
21
22 </TableLayout>
23
```

6. Click on the Graphical Layout Tab and you should see this:



Note that this view is very similar to a Visual Basic User Interface layout. Click back on the XML view to continue building.

7. Starting at line 21, insert the following Code to Create another TableRow and the displayText TextView object.

```
21
22  <TableRow
23          android:id="@+id/tableRow3"
24          android:layout_width="wrap_content"
25          android:layout_height="wrap_content" >
26
27          <TextView
28              android:id="@+id/displayText"
29              android:layout_width="wrap_content"
30              android:layout_height="wrap_content"
31              android:layout_span = "2"
32              android:text="---" />
33
34      </TableRow>
35
```

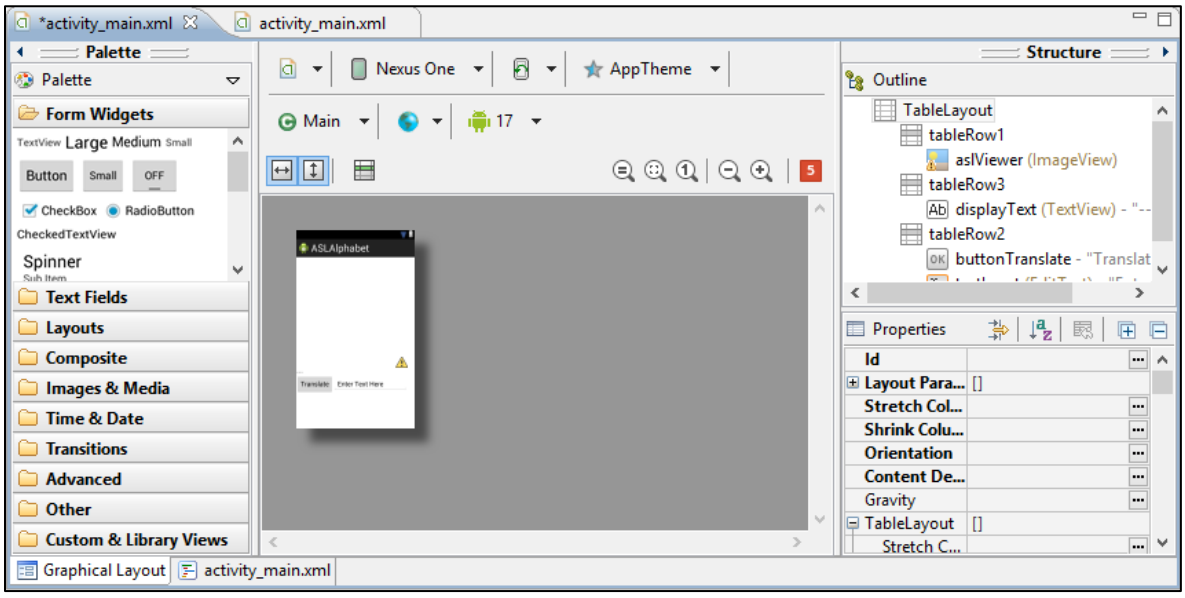
8. Type the following to create the Translate Button and EditText Objects:

```
36      <TableRow
37          android:id="@+id/tableRow2"
38          android:layout_width="wrap_content"
39          android:layout_height="wrap_content" >
40
41          <Button
42              android:id="@+id/buttonTranslate"
43              android:layout_width="wrap_content"
44              android:layout_height="wrap_content"
45              android:onClick = "setString"
46              android:text="Translate" />
47
48          <EditText
49              android:id="@+id/textInput"
50              android:layout_width="200dp"
51              android:layout_height="wrap_content"
52              android:ems="10"
53              android:text = "Enter Text Here" />
54
55      </TableRow>
```

9. Make sure Line 57 has the </TableLayout> Tag:

```
56
57  </TableLayout>
58
```

10. Click on 'Graphical Layout' and your User Interface should look like this:



11. You might see some yellow exclamation points. These indicate warnings that Android prefers you use @String resources to encode strings like object display texts. For now, we will Hard Code the strings within the XML. The advantage of creating a resource and using @Strings is that in App development this allows programmers to change strings (for translations or other preferences) by changing the @String xml file instead of hunting through a large collection of XML interfaces that might be created during the development of the App. For the purposes of these One Activity type tutorials, we will sometimes use Hard Coded strings.

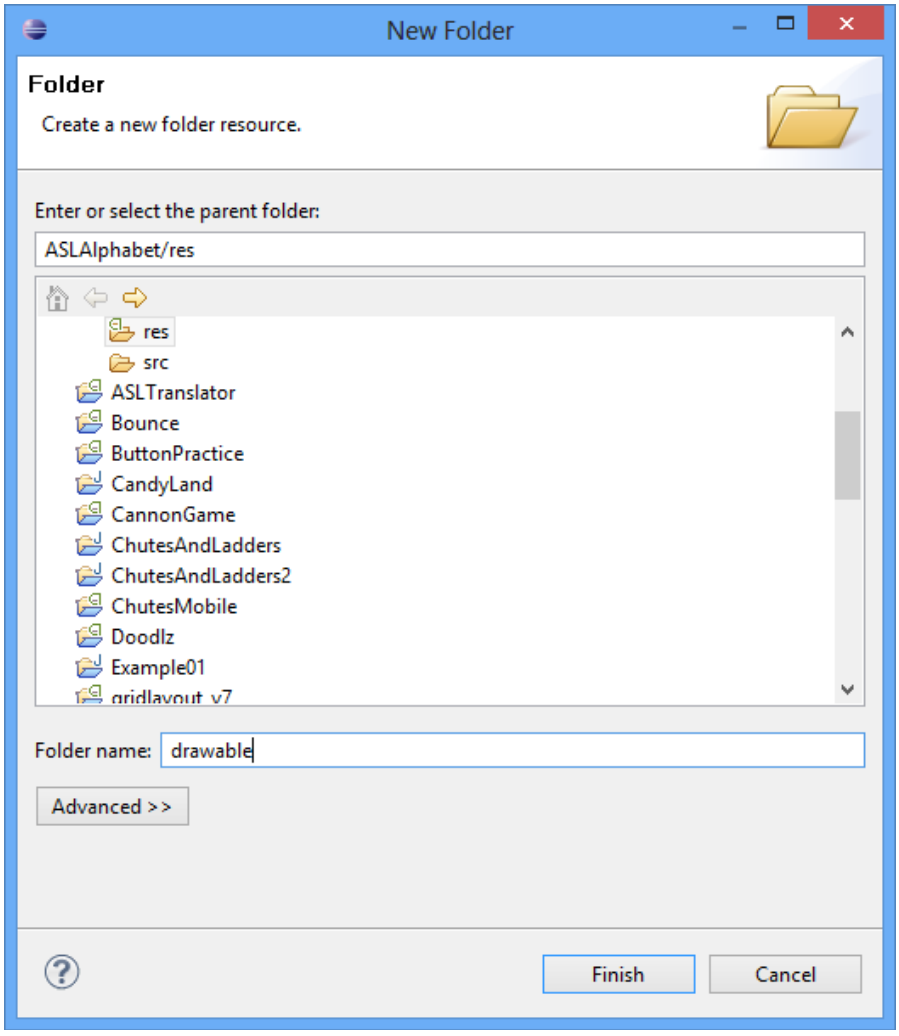
Phase 3: Create the Drawable Folder in Resources and import the images

Here we will import the .gif images representing the Sign Language for the Alphabet.

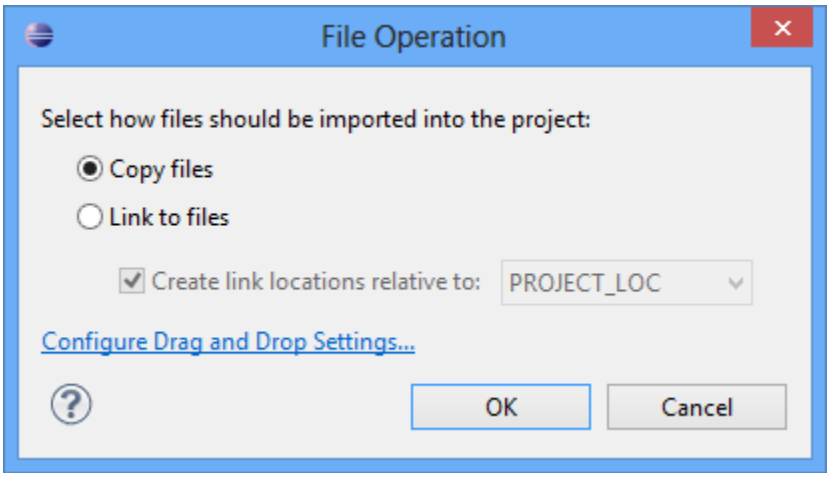
1. Go to the following URL:

   http://198.211.103.19/~nebomusic/aslalphabet.zip

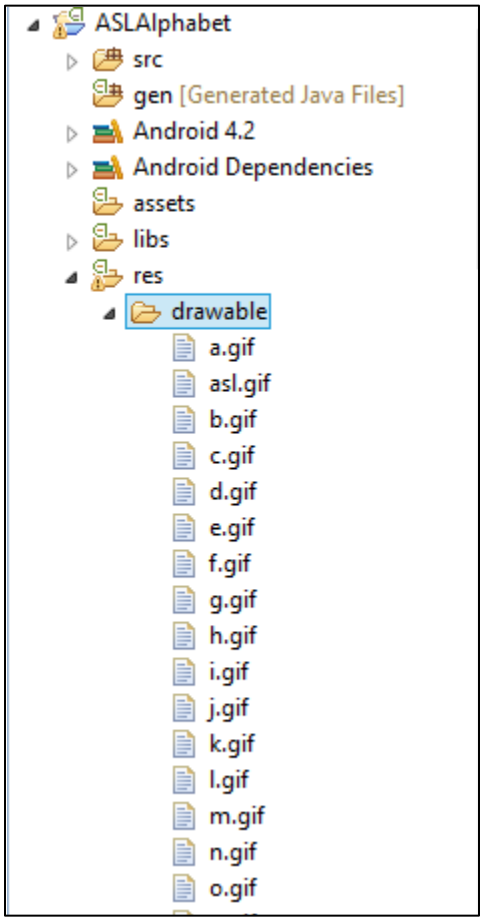
2. Download the above Zip File and Extract to Your Desktop. These are the .gif images for the ASL Alphabet.
3. Right Click on the 'res' folder in the Eclipse Project. Select New->Folder and name it 'drawable'



4. Go to the Folder where you have the ASL images and drag the images to the drawable folder. Select ‘Copy’ when prompted.



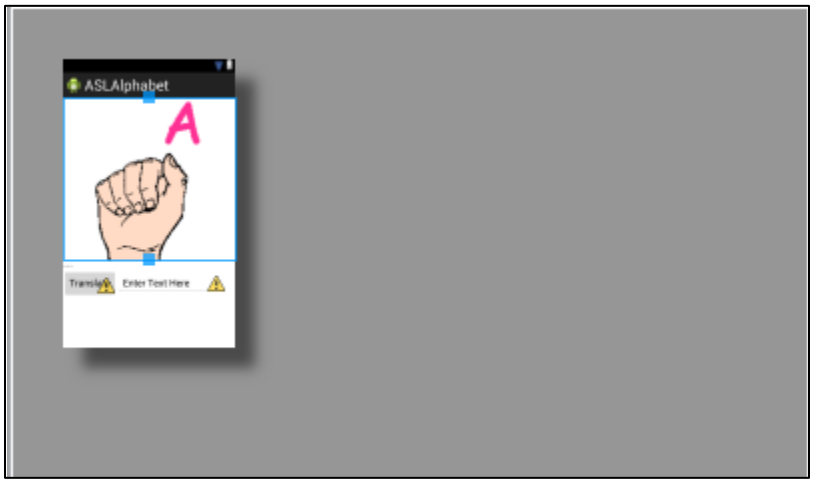
5. Expand the drawable folder by clicking on the Triangle, you should see the graphics:



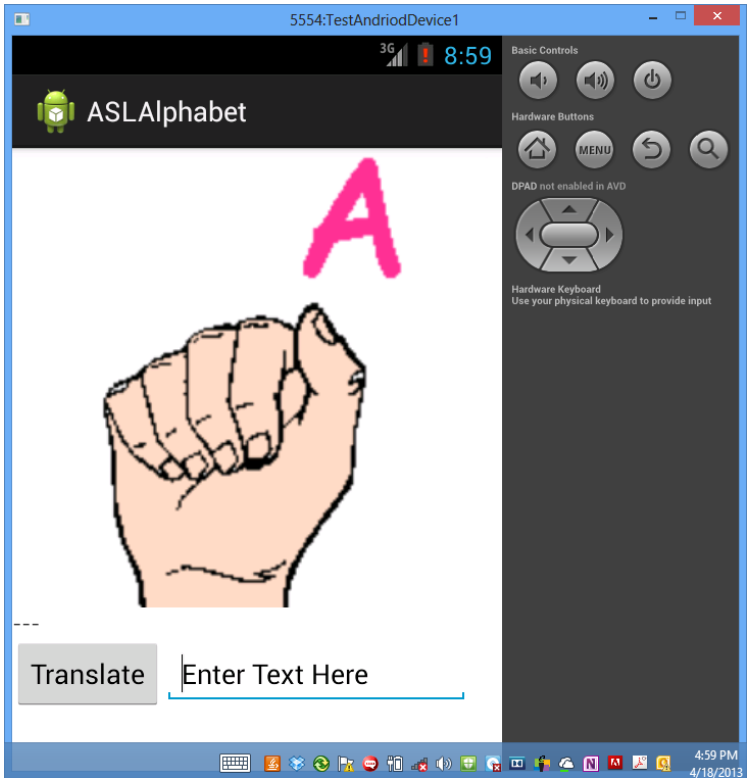
6. We will now modify the XML so the Letter A shows by default. Add the following line to the activity_main.xml code: (Line 18: android:src="@drawable/a" />)

```
12          <ImageView
13              android:id="@+id/aslViewer"
14              android:layout_width="wrap_content"
15              android:layout_height="300dp"
16              android:layout_span = "2"
17              android:onClick = "translateLetter"
18              android:src="@drawable/a" />
19              />
```

7. Click the Graphical Layout Tab and you should see this:



8. With the XML Code and User Interface set and the .gif image resources imported, we may now test the running of the App in the Emulator or Phone. Click the Play button and select “Android Application.” The App should launch on the phone.



9. However, if you click the Translate button or the Letter A – the App will crash. Why? Because we have not coded the Main Class. In the next phase we will write the functions for the Main Class.

Phase 4: Write the Code for the Main Class

We will now develop the Code for the Main Class. In this example the Control, Model, and View will all be handled within the Main. Recall the Class Diagram:

Class Main Extends Activity

Fields:

Android OS Objects:
Button translate: Calls the translate function when pressed
EditText enterText: Field where User enters phrase to translate
TextView displayText: Displays Text during run of Translation
ImageView aslImages: Displays the Images during run of Translation

Data for Translation Function
Int phraseIndex: Used to track location in Translation String
String letters: Message to be translated
String display: Holds the letters to be displayed in the displayText Object

Array char letterIndex: Stores in Hardcode the alphabet in lower case
Array int aslPics: Stores references to the R.drawable images for the ASL Alphabet.

Note that letterIndex and aslPics are parallel Arrays.

OnCreate Function (Acts similar to a Constructor in Android Activities)

super.onCreate
setContentView to main activity layout

Bind the Android Object instances to the layout objects
Translate -> R.id.buttonTranslate
enterText -> R.id.textInput
displayTest -> R.id.displayText
aslImages -> R.id.aslViewer

enterText.setSelectAllOnFocus(true) -> Selects all text in enterText

setString Function (Moves Text from enterText to letters string)

(Triggered by Touch Event on translate Button Object)
Hide Keyboard
Set phraseIndex to 0
Set display String to ""
Get the Input Text
Set Input Text to letters String

translateLetter Function(walks through letters string and displays images)

Check if EditText phrase was translated
Fetch the Current Letter according to phraseIndex
Display the ASL image corresponding to the Current Letter index
Add the current letter to the display String
Set the displayText to the display String

Check to see if you reach the end of the phrase and reset the phrase at end.

1. Android Code needs to import several libraries to drive the User Interface Objects. Type the following include statements in your Main Class: (Lines 1 to 14)

```
1  package com.marist.aslalphabet;
2
3  import android.os.Bundle;
4  import android.app.Activity;
5  import android.app.Service;
6  import android.text.Editable;
7  import android.view.Menu;
8  import android.view.View;
9  import android.view.inputmethod.InputMethodManager;
10 import android.widget.Button;
11 import android.widget.EditText;
12 import android.widget.ImageView;
13 import android.widget.TextView;
14
```

2. Next we will begin the Field definitions. Line 15 declares the public class Main and Lines 16 through 20 declare the User Interface objects.

```
15 public class Main extends Activity {
16     // View Objects
17     Button translate;
18     EditText enterText;
19     TextView displayText;
20     ImageView aslImages;
21
```

3. Lines 21 through 26 declare the fields we will use to run the Translation functions and display images and text.

```
21
22     // Variables for Translation and Display
23     int phraseIndex = 0;  // Keep track of the array indexes
24     String letters;       // Message to be Translated
25     String display;       // Will hold the letters already displayed and show them
26
```

4. Lines 26 through 25 declare a char array that will hold the letters of the alphabet plus the 'space' character.

```
26
27      // Array Libraries For Characters and Image References
28      char letterIndex [] = {'a', 'b', 'c', 'd',
29                             'e', 'f', 'g', 'h',
30                             'i', 'j', 'k', 'l',
31                             'm', 'n', 'o', 'p',
32                             'q', 'r', 's', 't',
33                             'u', 'v', 'w', 'x',
34                             'y', 'z', ' '};
35
```

5. Lines 35 to 44 declare an int array that will hold the references to the image resources located in the 'drawable' folder. Note that as you type this Array, Eclipse will help you auto-fill these values.

```
35
36      // American Sign Language image resources
37      int aslPics [] = {R.drawable.a, R.drawable.b, R.drawable.c, R.drawable.d,
38                        R.drawable.e, R.drawable.f, R.drawable.g, R.drawable.h,
39                        R.drawable.i, R.drawable.j, R.drawable.k, R.drawable.l,
40                        R.drawable.m, R.drawable.n, R.drawable.o, R.drawable.p,
41                        R.drawable.q, R.drawable.r, R.drawable.s, R.drawable.t,
42                        R.drawable.u, R.drawable.v, R.drawable.w, R.drawable.x,
43                        R.drawable.y, R.drawable.z, R.drawable.space};
44
```

6. A computer science and Java conventions note: In this example we are 'hard coding' the data referencing the alphabet characters and image resources. As Apps grow larger and more complicated with a wide range of data sets, data will be written or stored as files and then referred to within the Resources system. Many apps store data remotely on web servers to save space on the device. For this learning example, we will use hard coded data.

7. Write the Code for the onCreate Funtion (Lines 45 to 61).

```
45
46      @Override
47      protected void onCreate(Bundle savedInstanceState) {
48          super.onCreate(savedInstanceState);
49          setContentView(R.layout.activity_main);
50
51          // Attach objects to View Objects
52          translate = (Button) findViewById(R.id.buttonTranslate);
53          enterText = (EditText) findViewById(R.id.textInput);
54          displayText = (TextView) findViewById(R.id.displayText);
55          aslImages = (ImageView) findViewById(R.id.aslViewer);
56
57          // Select all the Text in the enterText Field
58          enterText.setSelectAllOnFocus(true);
59
60      } // End Function onCreate()
61
```

The 'onCreate' function serves as the Constructor within an Android Activity. The 'onCreate' is called when an activity is spawned or started. The purpose of the 'onCreate' is to connect the code objects in the Fields to the XML objects in the user interface. In addition, other fields can have their values set as needed similar to the role of a Constructor. A common coding error is to forget to 'attach' the objects to the XML views with the 'onCreate.' Again, this type of error will cause the App to Crash, but not be displayed in an error message. Always double check your 'onCreate.'

8. Write the code for the Menu inflator:

```
61
62      @Override
63      public boolean onCreateOptionsMenu(Menu menu) {
64          // Inflate the menu; this adds items to the action bar if it is present.
65          getMenuInflater().inflate(R.menu.activity_main, menu);
66          return true;
67      }
68
```

This is required by Android. In this App we will not use this function. However, when Apps have a menu option, this is the function that is called to bring up the menu. (You will usually see this with 'settings' in Apps).

9. Write the setString() function to get the Values from the enterText object and convert them to a lower cased String for the translateLetter function. (Lines 69 to 91)

```
69      // setString Function
70      // Acts when the Translate Button is pressed
71      public void setString(View v) {
72          // Set the text of displayText to '---'"
73          displayText.setText("---");
74
75          // Close Keyboard on lost focus
76          InputMethodManager imm = (InputMethodManager)this.getSystemService(Service.INPUT_METHOD_SERVICE);
77          imm.hideSoftInputFromWindow(v.getWindowToken(), 0);
78
79          // Reset the Phrase Index
80          phraseIndex = 0;
81          display = "";
82
83          // Get the Input Text
84          Editable input = enterText.getText();
85
86          // Convert to a string
87          String phrase = input.toString();
88          letters = phrase.toLowerCase();
89
90      } // End Function setString
91
```

Several interesting Android objects are presented here:

a. The “InputMethodManager” instance ‘imm’ (Lines 76 and 77) will handle the onscreen keyboard of the App. In this context the imm instance will close the on screen keyboard when the focus (touch) shifts to another object on the screen.
b. The “Editable” instance ‘input’ acts as a ‘go between.’ (Line 84) ‘input’ gathers the text from the enterText object and passes it to a temporary String object ‘phrase’.
c. We use the String function ‘toLowerCase()’ in line 88 to convert the text to lower case letters and pass this value to the letters String object.

10. Begin writing the translateLetter() function: (Lines 91 to 101)

```
91
92      // Displays Image when ImageView Touched.
93      // Also displays text of letters already translated
94      public void translateLetter(View v){
95
96          // Checks if letters string is null - displays message
97          // If the phrase has not been converted to a string
98          if (letters == null) {
99              displayText.setText("Press the Translate Button");
100         }
101
```

Lines 98 through 99 check the contents of the displayText object. If the displayText object is empty, a message appears that states “Press the Translate Button.”

11. Continue writing the translaterLetter() function: (Lines 101 to 119)

```
101
102         // Checks if letters sting is null - will not display
103         // ASL letters until Translate Button is pressed
104         if (letters != null) {
105
106             // Fetch the current character in the phrase
107             char currentLetter = letters.charAt(phraseIndex);
108
109             // add the letter to the display string
110             display += currentLetter;
111
112             // Search for the corresponding ASL image by Index
113             for (int i = 0; i < letterIndex.length; i++){
114                 if (letterIndex[i] == currentLetter) {
115                     // Display the image
116                     aslImages.setImageResource(aslPics[i]);
117                 } // end if
118             } // end for
119
```

Note the comments describe the purpose of the code. We are:

Retrieving the index from the letters Array of the currentLetter

Adding the letter to the display string

Retrieving the ASL image and setting the Image to the resource from the aslPics Array.

12. Finish the code for the Main Class: (Lines 119 to 136)

```
119
120             // Set the text to display the letters translated
121             displayText.setText(display);
122
123             // Advance to the Next Letter in the Phrase
124             phraseIndex++;
125
126             // Check to see if you reach the end of the phrase
127             if (phraseIndex > letters.length()-1) {
128                 // Reset back to the first character
129                 phraseIndex = 0;
130                 display = "";
131             } // end if
132         } // end Function translateLetter()
133     } // end if
134
135 } // End Class Main
136
```

13. Save, compile and Run the App on Your Device or Emulator.

Project Scoring Options: (Maximum 100 Points)

Complete the Directions and create a working ASL Alphabet Application: 85 Points
(Upload the Source Code and the apk package to your SkyDrive Folder)
(Upload a Jing demonstrating you running the App on your Emulator)

Modify the XML Code to create a more visually pleasing User Interface: Up to 8 Points
Changing the Background Colors, Button Size and Color, or any other creative touch to the user interface

Create a list containing common words or phrases and adding a button to show the phrase in the ImageView: Up to 10 Points