MOBILE DEVICES – ASSIGNMENT 1

Mobile Devices – Assignment 1

Submitted: October 17, 2014
By: Devan Shah 100428864
Submitted to: Michael Chang

Design Techniques

For assignment 1 I have constructed the mobile application calculator in Andriod using eclipse. Using the interface builder in eclipse for Andriod I was able to construct the UI design of the Andriod calculator. I came up with the sample design based on research on different types of calculators that already exists for different platforms and came to the conclusion on a UI to construct. I looked at calculators that exist for windows 7, iOS default calculator and default Andriod calculator, from the research I concluded to base the UI calculator based off windows 7 calculator with major changes in the design. Before going forward and constructing the design in eclipse I constructed a sample design sketch to see how the calculator would look on a phone. I also added more functionality then required by assignment 1 instructions and also support landscape orientation. Also when the orientation is changed it will preserve the data and restore the orientation based of the preserved data. Figure 1 shows the UI design of the android calculator in portrait view and Figure 2 shows the UI design of the android calculator in landscape view. The reason that the design are different is because the width and the height change when orientation change, therefore there is a need of reordering the buttons to make sure that all the buttons are visible on the screen.

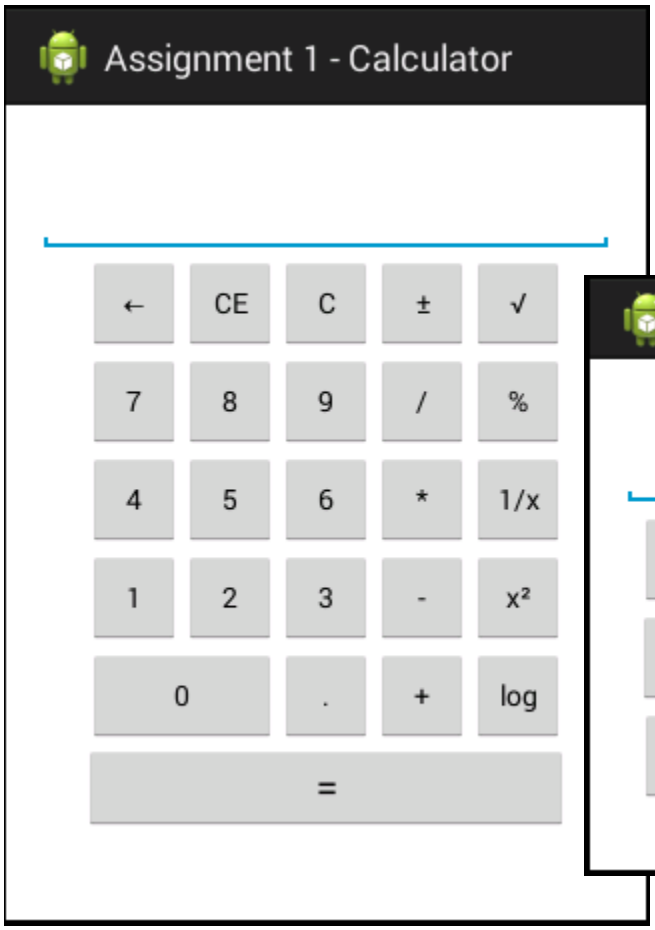

Figure 2 Andriod Calculator UI Design Portrait view

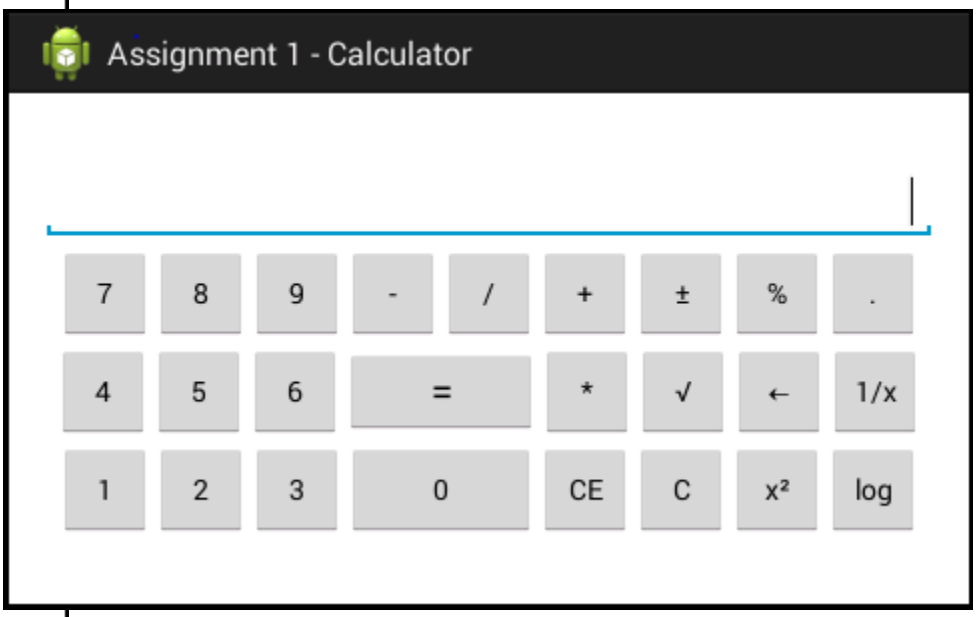

Figure 1Andriod Calculator UI Design LandScape View

Implementation Techniques

The Andriod calculator was implemented in eclipse using the android application development suite and the use of java. Java played a crucial role in implementing the calculator and having all the calculations performed with the use of build in java libraries and simple java arithmetic operations. I have implemented a wide range of functions aside from the basic arithmetic operators. The special functions include: plus/minus, Clear Entry (CE), Clear (C), Backspace (←), square root, percentage (%), one over x (1/x), squared (x^2) and log base 10 (log). The normal usage of the calculator still exists. During the implementation of the calculator memory was considered, and for that reason I made it so that my MainActivity class implemented View.OnClickListener which allows to interact with the buttons more efficiently. Figure 3 show the implementation that was used for the class definition.

```
@SuppressLint("NewApi") public class MainActivity extends ActionBarActivity implements View.OnClickListener {

    // Define buttons that are used in the calculator
```

Figure 3MainActivity class implementing View.OnClickListener

With the use of the OnClickListener I was able to retrieve the button objects in the onCreate function and set a setOnClickListener for all the buttons that were used. Figure 4 and Figure 5 show a sample code snip of how this was accomplished.

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

    // Get the number buttons
    one    = ( Button ) findViewById ( R.id.one );
    two    = ( Button ) findViewById ( R.id.two );
    three  = ( Button ) findViewById ( R.id.three );
    four   = ( Button ) findViewById ( R.id.four );
    five   = ( Button ) findViewById ( R.id.five );
    six    = ( Button ) findViewById ( R.id.six );
    seven  = ( Button ) findViewById ( R.id.seven );
    eight  = ( Button ) findViewById ( R.id.eight );
    nine   = ( Button ) findViewById ( R.id.nine );

    // Get the simple operation buttons
    decimal  = ( Button ) findViewById ( R.id.decimal );
    add      = ( Button ) findViewById ( R.id.add );
    minus    = ( Button ) findViewById ( R.id.minus );
    multiply = ( Button ) findViewById ( R.id.multiply );
    divide   = ( Button ) findViewById ( R.id.divide );
    equal    = ( Button ) findViewById ( R.id.equal );
```

Figure 4Defining the buttons that are used for the calculator

```
// Define the click event for number buttons
one.setOnClickListener(this) ;
two.setOnClickListener(this) ;
three.setOnClickListener(this) ;
four.setOnClickListener(this) ;
five.setOnClickListener(this) ;
six.setOnClickListener(this) ;
seven.setOnClickListener(this) ;
eight.setOnClickListener(this) ;
nine.setOnClickListener(this) ;

// Define the click event for operation buttons
decimal.setOnClickListener(this) ;
add.setOnClickListener(this) ;
minus.setOnClickListener(this) ;
multiply.setOnClickListener(this) ;
divide.setOnClickListener(this) ;
equal.setOnClickListener(this) ;
```

Figure 5 Setting the "setOnClickListener" for each of the buttons to increase memory comsumption.

With the implementation of the View.OnClickListener in my project it helped to better organize the code and allow for an easy way to add more functionality to the calculator. The View.OnClickListener allows to use an shared onClick function to determine which action each of the buttons would do with the use of a switch statements based on the button that is pressed as the condition of the cases. Figures 6, 7 and 8 are an example of how this was accomplished.

```
@SuppressLint("NewApi") @Override
public void onClick ( View arg0 )
{
    Editable currentString = resultDisplay.
    String currentProgressString ;

    switch ( arg0.getId() )
    {
        case R.id.one:

            if ( number1Holder == null || r
```

Figure 6 Example of how the switch was defined on the onClick function to handle all the buttons

```
case R.id.clear:
    number1 = 0 ;
    number2 = 0 ;
    number1Holder = "";
    number2Holder = "";
    resultDisplay.setText ( "" ) ;
    progressLabel.setText ( "" ) ;
break;
case R.id.clear_entry:
    number2 = 0 ;
    number2Holder = "";
    resultDisplay.setText ( "" ) ;
    if ( !progressLabel.getText().t
        progressLabel.setText( prog
    }
```

Figure 7Further examples of how the switch was used to handle more functions

```
case R.id.equal:

    if ( ( ( number1Holder != null && !number1Holder.isEmpty() ) &&
              ( number2Holder != null && !number2Holder.isEmpty() ) )
         )
        {
           operation(Float.parseFloat(number1Holder),Float.parseFloat(number2Holder));
        }
        else if ( ( number1Holder != null && !number1Holder.isEmpty() ) && number1 != 0 )
        {
           operation(number1,Float.parseFloat(number1Holder));
        }
```

Figure 8Example of a full block of code in the switch to handle when the equal button is clicked

This structure in the code allows for easy access to make changes to the code to add more functions or to even fix issues that exists for specific buttons. All is in one place and the variables are global, there for there is only a need to set them once.

Following are some sample runs by performing some arithmetic operations and some complex operations.

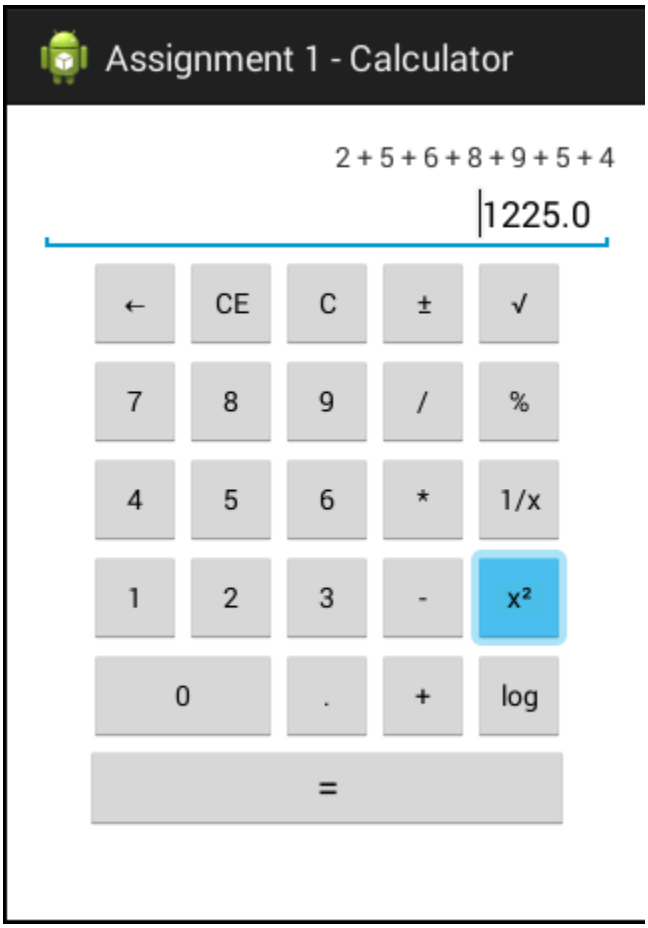

Figure 11 Shows performing calculations with multiple numbers and the results show up in a display at the top and the use of the sauared function

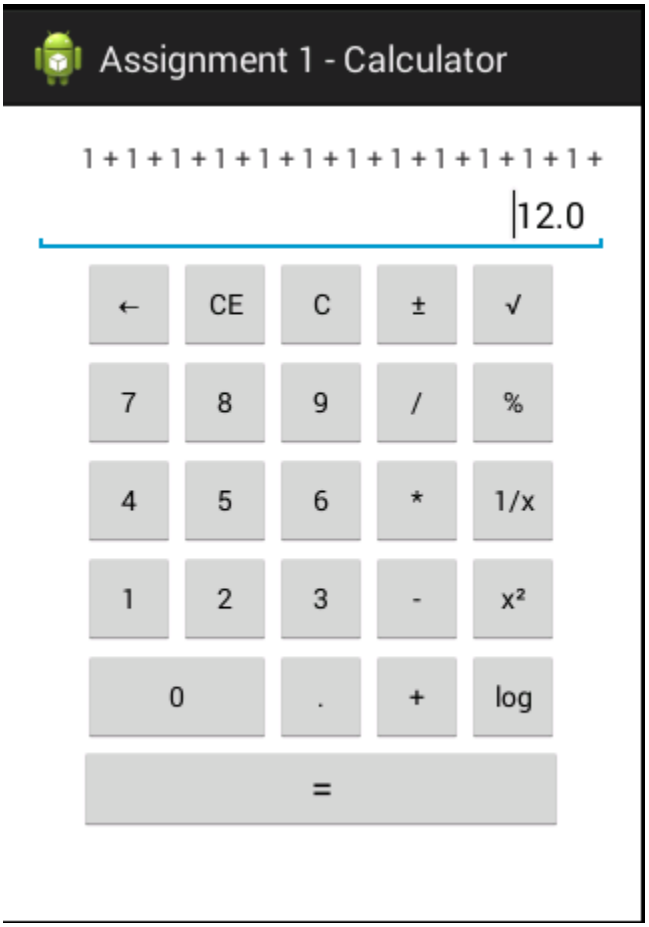

Figure 9Simple incremental addition example

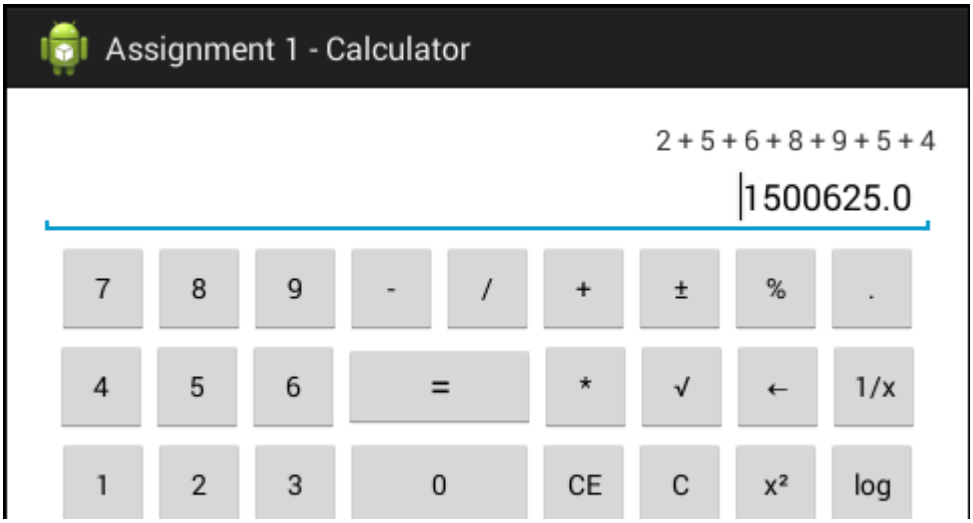

Figure 10Thie is the same calculation as Figure 10 but the orientation was changed and all the data was saved.