
UCD School of Computer Science

# Graphic User Interface (GUI)- Coding

*Dr. Abraham (Abey) Campbell*

# Objectives

- Coding using the MVC architecture
- Building the GUI and handle events programmatically
- Customize the GUI using a GridLayout and Layout Parameters

# Tic Tac Toe (1 of 8)

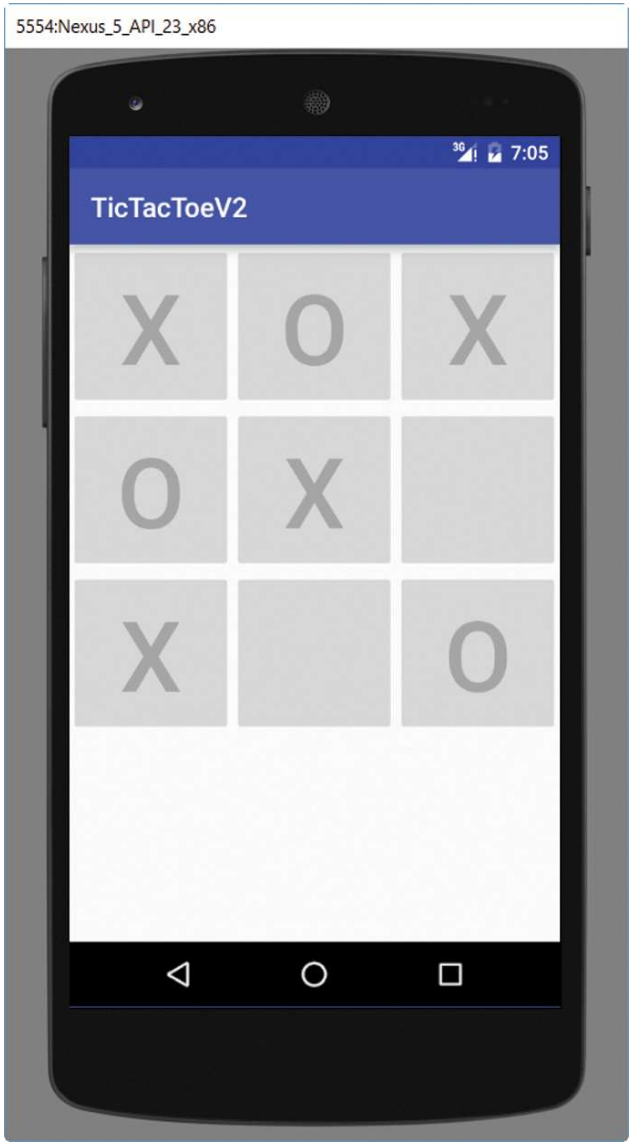
- More Model-View-Controller
- No XML layout: building a GUI by code
- GridLayout
- More Event Handling
- Layout Parameters
- AlertDialog

# Tic Tac Toe (2 of 8)

- Sometimes the GUI is dynamic and cannot be hard coded using an XML file.
- For example, the number of buttons may depend on data stored in a file or retrieved from a remote web site.
- In this case, the GUI needs to be created by code.

# Tic Tac Toe (3 of 8)

- Or maybe the GUI is such that it makes sense to do it by code.
- For a Tic Tac Toe app, we could edit an XML layout file and place nine buttons in it.
- But it is easier to have a 3 × 3 two-dimensional array of buttons that we can access by code.


5554:Nexus_5_API_23_x86
7:05
TicTacToeV2
X
O
X
O
X
X
O

# Tic Tac Toe (5 of 8)

- In this app, we do not use an XML file for the GUI.
- Instead, we do it by code, using a 3 × 3 two-dimensional array of buttons.
- We still use the Model-View-Controller architecture; we have two classes: one for the Model, one for the View and the Controller together (in the last version of the app, we will split the View and the Controller).

# Tic Tac Toe (6 of 8)

- The TicTacToe class represents the Model.
- It has two instance variables:
- A 3 × 3 two-dimensional array of ints.
- An int to store whose turn it is to play.

# Tic Tac Toe (7 of 8)

- An array value of 0 means the cell is available.
- An array value of 1 means that player 1 has played at that cell position (X for example).
- An array value of 2 means that player 2 has played at that cell position (O for example).
- Note that we could have used an array of chars instead ( ‘ ‘, ‘X’, and ‘O’).

# Tic Tac Toe (8 of 8)

- The TicTacToe class includes a number of methods to play the game, check if somebody won, check if the game is over, reset the game, …
- See Example for the TicTacToe class.

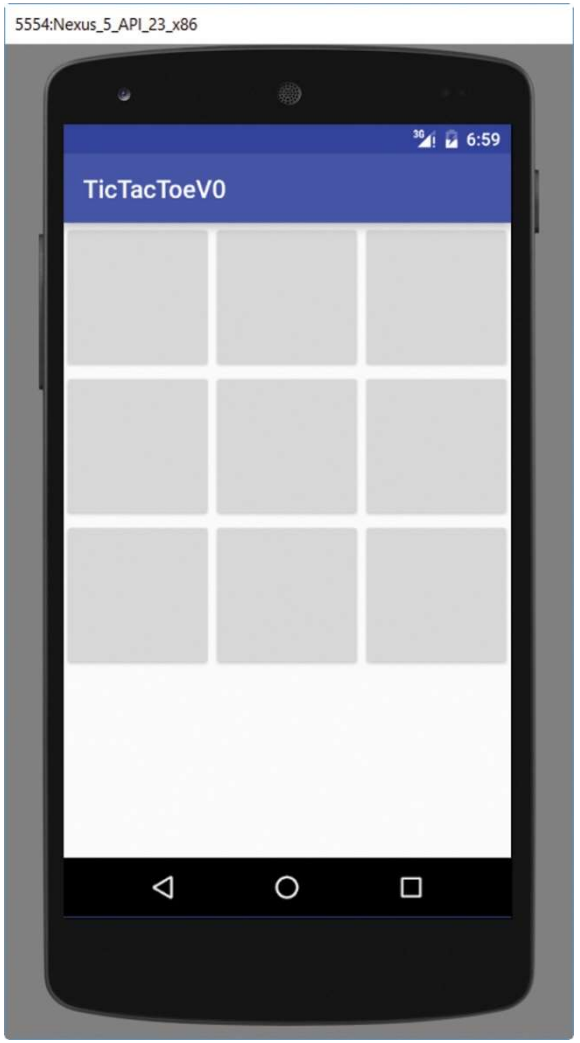
# Tic Tac Toe Class

- Viewing Code

```
public class TicTacToe {
    public static final int SIDE = 3;
    private int turn;
    private int [][] game;

    public TicTacToe( ) {
        game = new int[SIDE][SIDE];
        resetGame( );
    }

    public int play( int row, int col ) {
        int currentTurn = turn;
        if( row >= 0 && col >= 0 && row < SIDE && col < SIDE
                && game[row][col] == 0 ) {
            game[row][col] = turn;
            if( turn == 1 )
                turn = 2;
            else
                turn = 1;
            return currentTurn;
        }
        else
            return 0;
    }

    public int whoWon( ) {
        int rows = checkRows( );
        if ( rows > 0 )
            return rows;
        int columns = checkColumns( );
        if( columns > 0 )
            return columns;
        int diagonals = checkDiagonals( );
        if( diagonals > 0 )
            return diagonals;
        return 0;
    }

    protected int checkRows( ) {
        for( int row = 0; row < SIDE; row++ )
            if ( game[row][0] != 0 && game[row][0] == game[row][1]
                    && game[row][1] == game[row][2] )
                return game[row][0];
        return 0;
    }

    protected int checkColumns( ) {
        for( int col = 0; col < SIDE; col++ )
            if ( game[0][col] != 0 && game[0][col] == game[1][col]
                    && game[1][col] == game[2][col] )
                return game[0][col];
        return 0;
    }

    protected int checkDiagonals( ) {
        if ( game[0][0] != 0 && game[0][0] == game[1][1]
                && game[1][1] == game[2][2] )
            return game[0][0];
        if ( game[0][2] != 0 && game[0][2] == game[1][1]
                && game[1][1] == game[2][0] )
            return game[2][0];
        return 0;
    }

    public boolean canNotPlay( ) {
        boolean result = true;
        for (int row = 0; row < SIDE; row++)
            for( int col = 0; col < SIDE; col++ )
                if ( game[row][col] == 0 )
                    result = false;
        return result;
    }

    public boolean isGameOver( ) {
        return canNotPlay( ) || ( whoWon( ) > 0 );
    }

    public void resetGame( ) {
        for (int row = 0; row < SIDE; row++)
            for( int col = 0; col < SIDE; col++ )
                game[row][col] = 0;
        turn = 1;
    }
}
```

# Version 0

- In Version 0, we just display a grid of 9 Buttons organized as a 3 × 3 grid.



# MainActivity (1 of 6)

- We use a GridLayout to manage the screen and place the Buttons inside it.
- A GridLayout is a layout manager that places its children in a rectangular grid.

# MainActivity (2 of 6)

- Retrieve the width of the screen
- Define and instantiate a GridLayout with three rows and three columns
- Instantiate the 3 × 3 array of Buttons
- Add the nine Buttons to the layout
- Set the GridLayout as the layout manager of the view managed by this activity

# MainActivity (3 of 6)

- We want to have a 3 × 3 two-dimensional array of Buttons to match the functionality in the Model.

  // Instance variable buttons

  private Button [][] buttons;
- The Button class is in the android.widget package.

# MainActivity (4 of 6)

- We code the GUI in the buildGuiByCode method.
- We assume that the user will only play in vertical orientation.
- ➔ we can assume that the width of the screen is smaller than its height.

# MainActivity (5 of 6)

- We retrieve the width of the screen dynamically; each button's width and height is equal to one third of the screen's width.
- In this way, the GUI is consistent across various devices.

# MainActivity (6 of 6)

- The getWindowManager method of Activity returns a WindowManager object.
- The getDefaultDisplay method of the WindowManager class returns a Display object.
- The getSize method of the Display class enables us to retrieve the size of the display, i.e., the screen.

# Retrieving the Size of the Screen (1 of 2)

- We can chain the method calls:

```
Point size = new Point( );
getWindowManager( ).getDefaultDisplay( )
                                .getSize( size );
```

- getSize is a void method that sets the x and y public instance variable of its Point argument (here size).

# Retrieving the Size of the Screen (2 of 2)

- Now we can set the side of the Buttons:

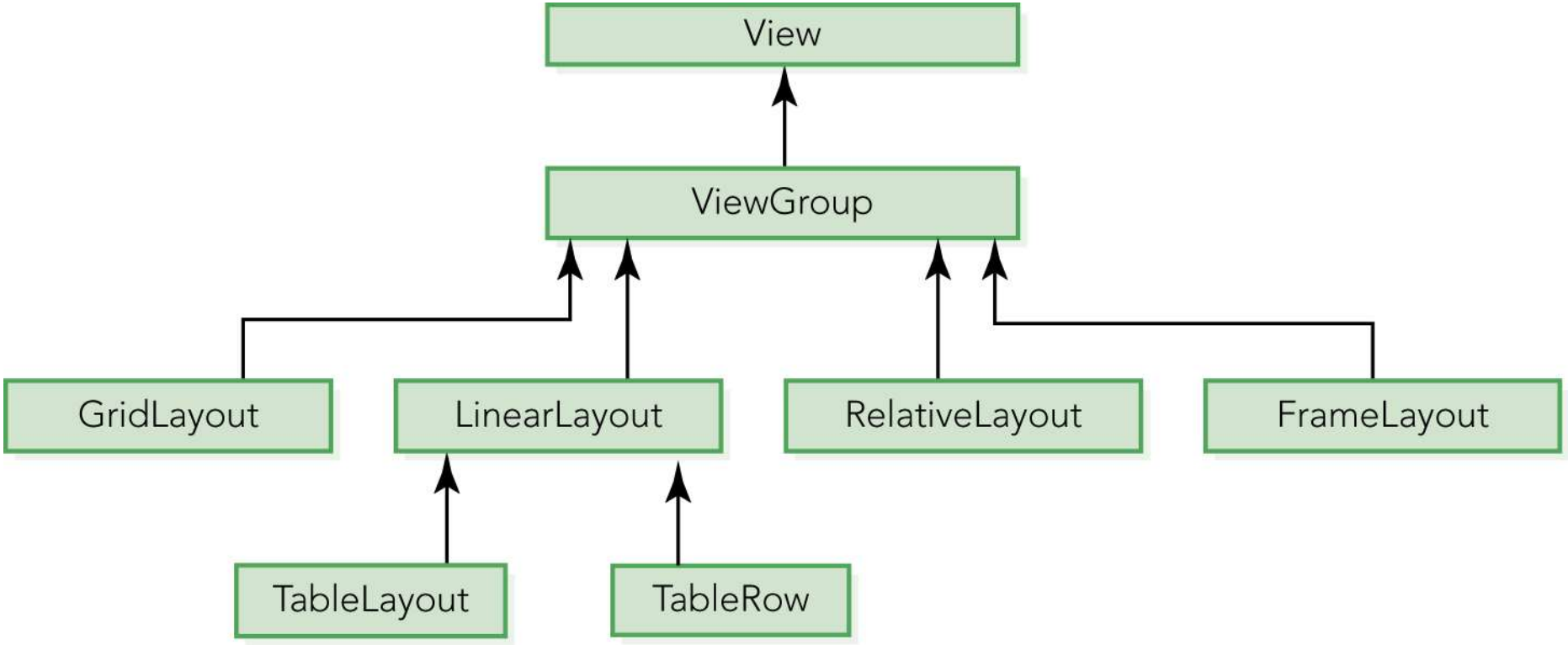
```
int w = size.x / TicTacToe.SIDE;
```

- Next, we can assign w to the width and height of each Button.

# Organizing the Layout

- The Android library provides us with many classes to manage various GUI layouts.
- The default layout is RelativeLayout.
- Here, we want to display a 3 × 3 grid of Buttons.
- The GridLayout class enables us to manage GUI components in a grid.

# Hierarchy of Layout Classes



# GridLayout (1 of 2)

- We use the following GridLayout constructor:

  GridLayout( Context context )

- MainActivity inherits from AppCompatActivity, which inherits from Activity, which inherits from Context. Therefore, a MainActivity “is a” Context ➔ we can pass *this* as the argument to the GridLayout constructor.

  GridLayout gridLayout = new GridLayout( this );

# GridLayout (2 of 2)

We use the setColumnCount and setRowCount methods of GridLayout to set the number of rows and columns:

```
gridLayout.setColumnCount( TicTacToe.SIDE );
gridLayout.setRowCount( TicTacToe.SIDE );
```

# Buttons

- After instantiating buttons, we use a double loop to instantiate each Button element and add it to gridLayout.
- We use the following Button constructor:

  Button( Context context )

- It is very similar to the GridLayout constructor.

# Button

- Again, we pass *this* to the constructor (a MainActivity “is a” Context):

```
buttons[row][col] = new Button( this );
```

- Now we need to add the current Button to gridLayout.

# Adding a View to a ViewGroup

- We use the addView method from ViewGroup:

  addView( View child, int w, int h )

- It adds the View child to the ViewGroup calling the addView method using width w and height h for child.

# GridLayout Methods

| Constructor | |
|---|---|
| GridLayout( Context context ) | Constructs a GridLayout within the app environment defined by context. |
| **Public Methods** | |
| setRowCount( int rows ) | Sets the number of rows in the grid to rows. |
| setColumnCount( int cols) | Sets the number of columns in the grid to cols. |
| addView( View child, int w, int h ) | Method inherited from ViewGroup; adds child to this ViewGroup using width w and height h. |

# Adding a Button to a GridLayout

- Button inherits from View: a Button “is a” View.
- GridLayout inherits from ViewGroup: a GridLayout “is a” ViewGroup.
- Thus, we can call addView with a GridLayout, passing a Button as the first argument:

```
gridLayout.addView( buttons[row][col], w, w );
```

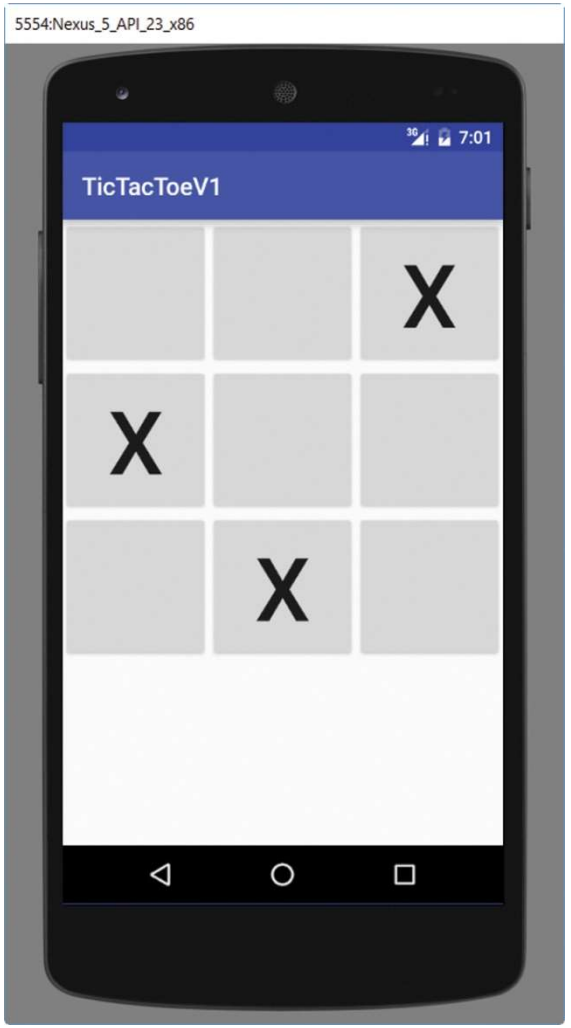
# Setting the View of the Activity

- Now that the View (gridLayout) is built, we can set it as the View for the Activity.
- We use the following method of the Activity class:

```
void setContentView( View v )
setContentView( gridLayout );
```

# Event Handling (1 of 9)

- In Version 1, we write an X inside a Button when the user clicks on it.
- We set up event handling by code.



# Event Handling (2 of 9)

- Handling an event involves three steps:
  - Write an event handler class ( a class that implements a listener interface).
  - Instantiate an object of that class.
  - Register that object on one or more GUI components.

# Event Handling (3 of 9)

- The type of event we want to capture determines the listener interface we need to implement.
- For a click event, we want to implement the View.OnClickListener interface (OnClickListener is a public static interface of the View class).
- It has one abstract method, onClick.

# Event Handling (4 of 9)

- We code an event handler class as a private inner class of MainActivity.
- It must overrides the onClick method, which has the following header:

  public abstract void onClick( View view )
- view is the View where the event originated.

# Event Handling (5 of 9)

```
private ButtonHandler implements
   View.OnClickListener {
  public void onClick( View view ) {
    Log.w( "MainActivity", "view = " + view );
    // handle the click here
  }
}
```

# Event Handling (6 of 9)

- Inside onClick, we loop through the array buttons and compare the current Button to View and therefore retrieve the row and column indexes of the Button that was clicked.
- We then call a method to update the View, passing the row and the column indexes.
- The update method updates the View.

# Event Handling (7 of 9)

```
public void onClick( View v ) {
 Log.w( "MainActivity", "Inside onClick, v = " + v );

 for( int row = 0; row < TicTacToe.SIDE; row ++ )
      for( int column = 0; column < TicTacToe.SIDE;
                                          column++ )
           if( v == buttons[row][column] )
              update( row, column );
}
```

# Event Handling (8 of 9)

- In Version 1, the update method simply outputs the row and column indexes to Logcat and writes an X inside the button that was clicked.

# Event Handling (9 of 9)

```
public void update( int row, int col ) {
    Log.w( "MainActivity", "Inside update: "
                + row + ", " + col );
     buttons[row][col].setText( "X" );
}
```

# Setting Up Event Handling

```
ButtonHandler bh = new ButtonHandler( );

for( int row = 0; row < TicTacToe.SIDE; row++ ) {
    for( int col = 0; col < TicTacToe.SIDE; col++ ) {
        …
        buttons[row][col].setOnClickListener( bh
);
        …
```

# Integrating the Model

- In Version 2, we integrate the Model to enable game play and enforce the rules.
- We add a TicTacToe instance variable to the MainActivity class; with it, we can call the various methods of the TicTacToe class to play the game and enforce the rules.

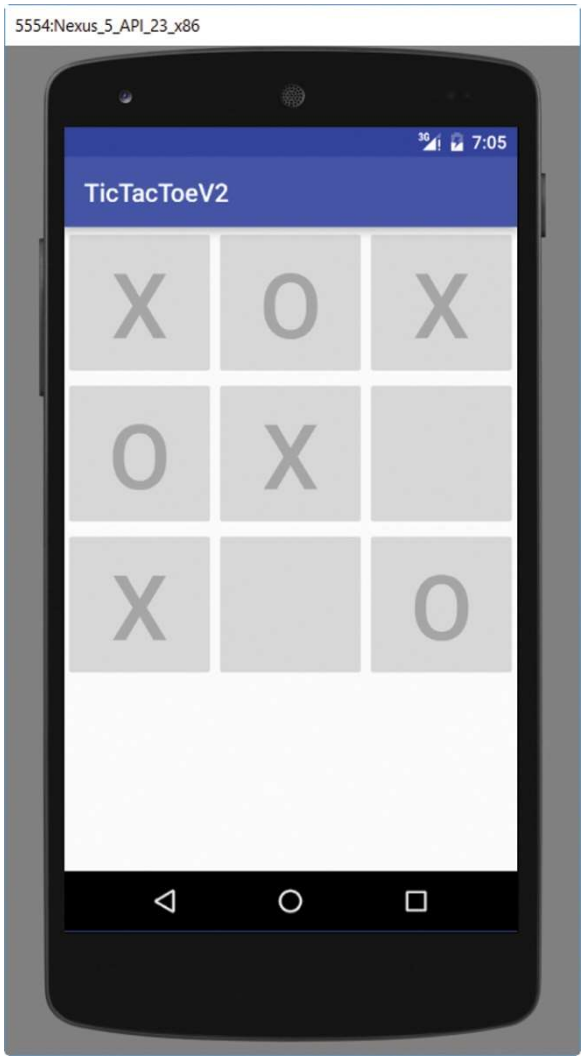
```
private TicTacToe tttGame;
```

- We instantiate it inside onCreate.

# Enabling Play

- Inside update, we call play with the TicTacToe object.
- Depending on what play returns, we write an X or an O in the button that was clicked, or do nothing if the button was clicked earlier.

# Version 2



# Enabling Play (1 of 3)

- Inside update, we also check whether the game is over; if it is, we disable all the buttons.
- For this, we add the enableButtons method.

```
public void enableButtons( boolean enable )
```

- If enable is true, we enable all the Buttons; if it is false, we disable them.

# Enabling Play (2 of 3)

- The setEnabled method of the TextView class, inherited by the Button class, enables us to either enable or disable a Button.

  public void setEnabled( boolean enabled )

- If enabled is true, the Button calling the method is enabled; if enabled is false, the Button calling the method is disabled.

# Enabling Play (3 of 3)

```
public void enableButtons( boolean enabled ) {
    for( int row = 0; row < TicTacToe.SIDE; row++ )
      for( int col = 0; col < TicTacToe.SIDE; col++ )
        buttons[row][col].setEnabled( enabled );
}
```

# update Method (1 of 2)

```
public void update( int row, int col ) {
    // play
    // check whether the game is over
}
```

# update Method (2 of 2)

```
public void update( int row, int col ) {
    int play = tttGame.play( row, col );
    if( play == 1 )
      buttons[row][col].setText( "X" );
    else if( play == 2 )
      buttons[row][col].setText( "O" );
    if( tttGame.isGameOver( ) ) // game over
      enableButtons( false );
}
```

# Inner classes (1 of 3)

- In Version 3, we add a TextView below the buttons to show the status of the game.
- We use the GridLayout.LayoutParams class in order to set the layout parameters for a View that we want to add to a GridLayout.


5554:Nexus_5_API_23_x86
7:08
TicTacToeV3
O
X
X
O
PLAY !!

# Inner classes (3 of 3)

- The notation GridLayout.LayoutParams denotes that LayoutParams is a public static inner class of GridLayout.
- Many layout classes (GridLayout, RelativeLayout, LinearLayout, ..) have public static inner classes that provide functionality to set layout parameters for a View when adding it.

# B Is a Public Static Inner Class of A (1 of 2)

```
public class A {
  // some code for class A
  public static class B {
    // some code for class B
  }
}
```

# B Is a Public Static Inner Class of A (2 of 2)

- In a client class, we refer to B as A.B

  A.B b = new A.B( );

# Placing the TextView in the GridLayout (1 of 2)

- We place the TextView in the fourth row of the GridLayout, using the whole row for it.
- The GridLayout.LayoutParams class (LayoutParams is a public static inner class of GridLayout) enables us to define a rectangle of cells within the grid so that we can place a GUI component there.

# Placing the TextView in the GridLayout (2 of 2)

- The GridLayout class includes the spec method; it returns a Spec object (Spec is also a public static inner class of GridLayout) and takes 2 parameters; they specify a starting index and a size (number of cells).

public static GridLayout.Spec spec( int start, int size )

# Creating a GridLayout.Spec

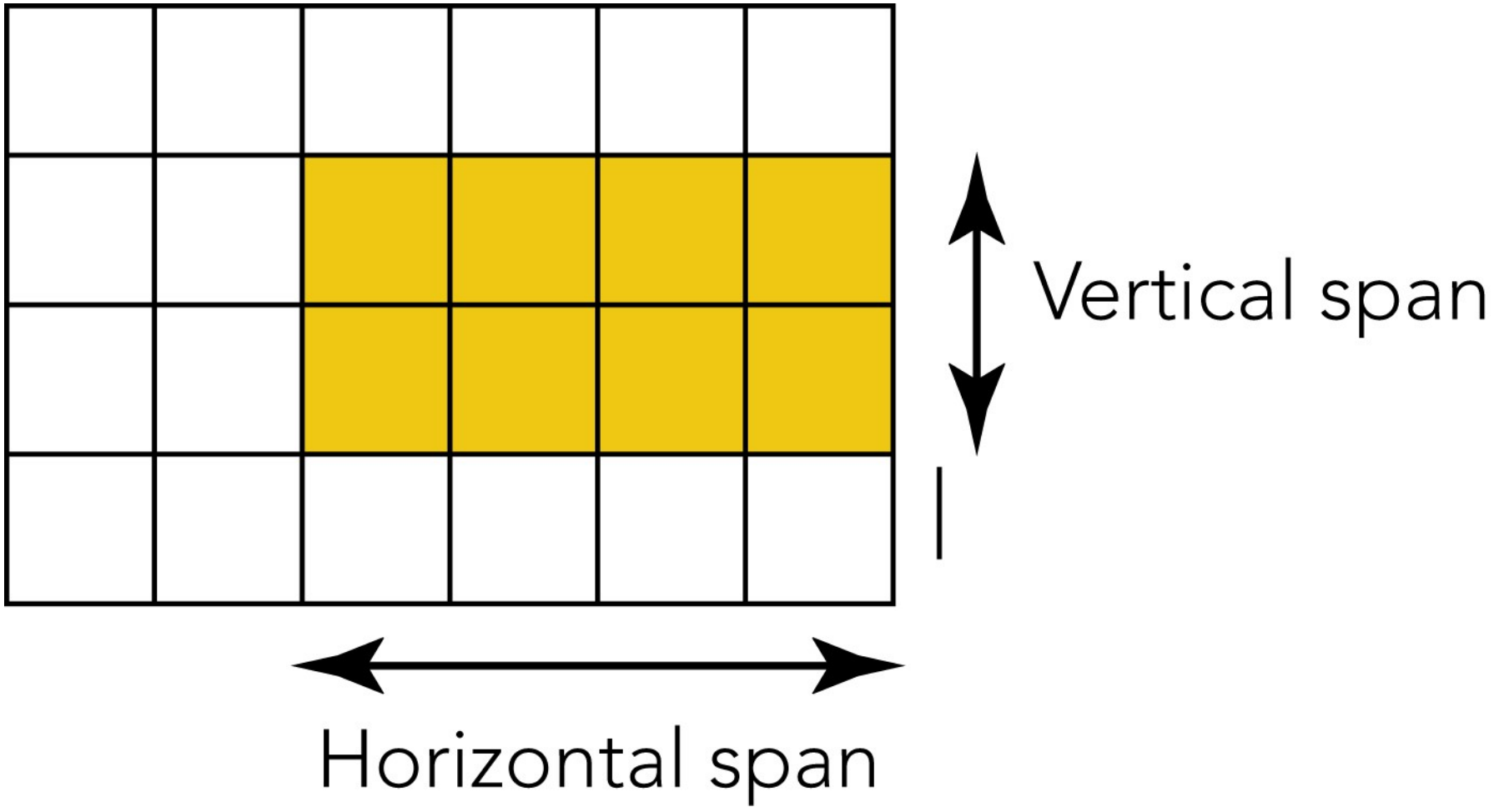
- Create a vertical span that starts at row 1 and spans 2 rows:

  GridLayout.Spec rowSpec =
  GridLayout.spec( 1, 2 );

- Create a horizontal span that starts at column 2 and spans 4 columns:

  GridLayout.Spec columnSpec =
  GridLayout.spec( 2, 4 );

# Visualizing the GridLayout.Spec



# Creating a GridLayout.LayoutParams

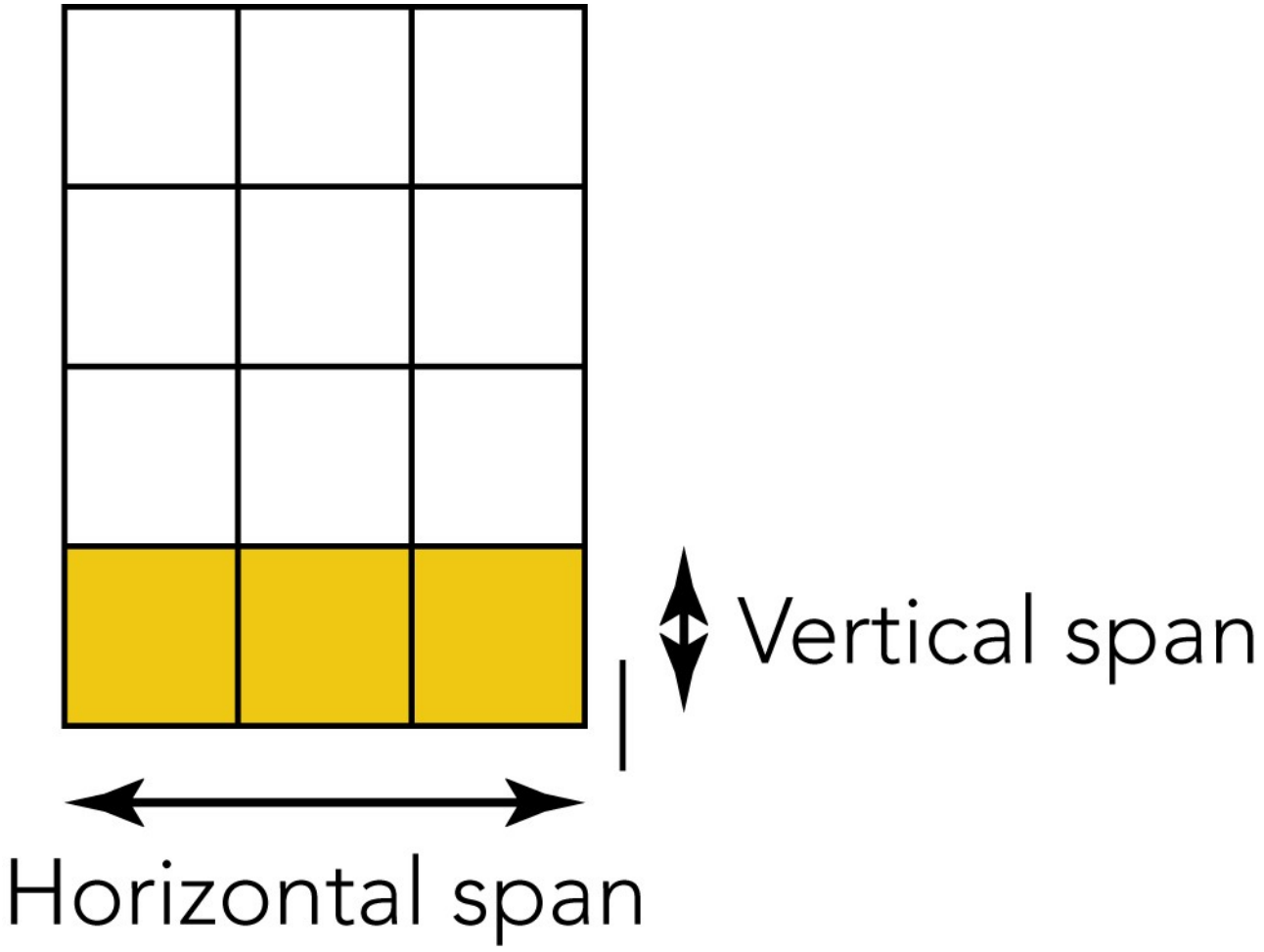
- We can use two GridLayout.Spec objects to create a GridLayout.LayoutParams object that we will use to position the TextView within the GridLayout.

```
GridLayout.LayoutParams lp = new
    GridLayout.LayoutParams( rowSpec,
    columnSpec );
```

# Creating a GridLayout.LayoutParams

- The first argument of the GridLayout.LayoutParams constructors specifies the vertical span.
- The second argument of the GridLayout.LayoutParams constructors specifies the horizontal span.

# Here, We Want This Position



# Modifying the GridLayout

- The GridLayout now has an extra row.

```
GridLayout gridLayout = new GridLayout( this );
gridLayout.setColumnCount( TicTacToe.SIDE );
gridLayout.setRowCount( TicTacToe.SIDE + 1 );
```

# Creating a GridLayout.LayoutParams (1 of 2)

```
GridLayout.Spec rowSpec =
   GridLayout.spec( 3, 1 );
GridLayout.Spec columnSpec =
   GridLayout.spec( 0, 3 );
GridLayout.LayoutParams lp = new
   GridLayout.LayoutParams( rowSpec,
   columnSpec );
```

# Creating a GridLayout.LayoutParams (2 of 2)

```
status = new TextView( this );
status.setLayoutParams( lp );
// now set status's properties
// width, height, color
// text, text size, ..
```

# Setting Properties of the TextView

- Now we can use the setBackgroundColor (inherited from View) and setWidth, setHeight, setGravity, setTextSize and setText methods from the TextView class to set the properties and content of the TextView.

  ```
  status.setBackgroundColor( Color.GREEN );
  ...
  ```

- See Example 3.8

# Adding the TextView

- Add the textView to the GridLayout:

  gridLayout.addView( status );

# Updating the View

- Inside the update method, if the game is over
  (if tttGame.isOver( ) )
- Change the color to red:

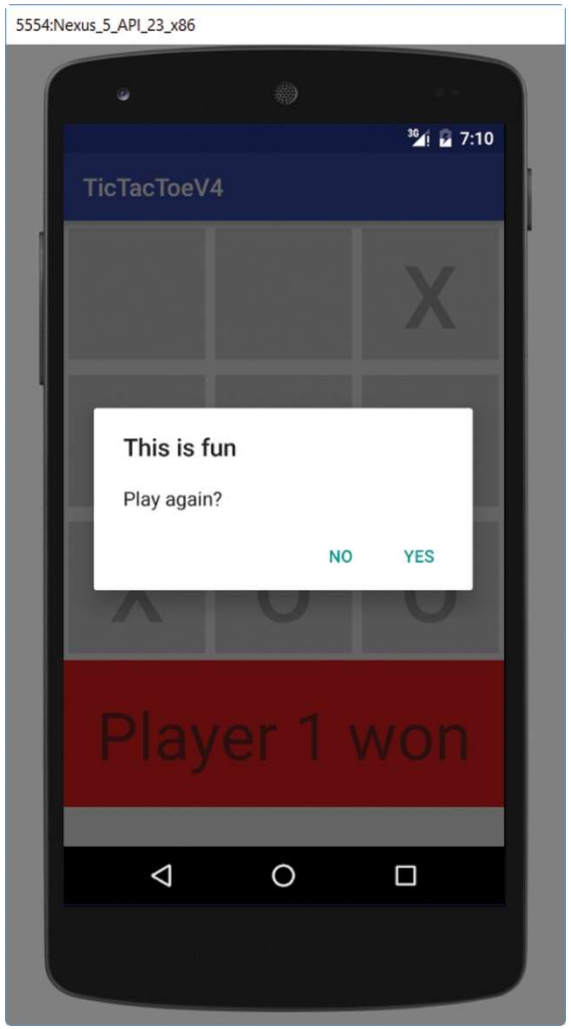
  status.setBackgroundColor( Color.RED );
- Ask the Model and set the text of status accordingly:

  status.setText( tttGame.result( );

# Play Again? (1 of 3)

- In Version 4, we ask the uses whether he/she wants to play again after the game is over.
- When the game is over, we pop up a dialog box.

# Play Again? (2 of 3)



# Play Again? (3 of 3)

- We use the AlertDialog.Builder class to pop up a dialog box (Builder is a public static inner class of AlertDialog)
- It contains three buttons, which we can set with the setPositiveButton, setNeutralButton, and setNegativeButton methods.

# AlertDialog.Builder Class (1 of 4)

- First we instantiate an AlertDialog.Builder object using the constructor:

```
public AlertDialog.Builder ( Context context )
AlertDialog.Builder alert = new
AlertDialog.Builder( this );
```

- Then we call the set…Button methods.

# AlertDialog.Builder Class (2 of 4)

```
public AlertDialog.Builder setPositiveButton(
    CharSequence text,
    DialogInterface.OnClickListener listener )
```

- CharSequence is an interface that is implemented by the String class.
- Therefore, a String "is a" CharSequence and can be used as the first argument.

# AlertDialog.Builder Class (3 of 4)

- DialogInterface.OnClickListener is an interface with a single method, onClick.
- OnClickListener is a public static inner interface of the DialogInterface interface.
- The onClick method is automatically clicked after the user selects one of the buttons of the dialog box.

# AlertDialog.Builder Class (4 of 4)

- We need to code a class (PlayDialog) that implements the DialogInterface.OnClickListener interface.
- We code the onClick method: inside, we either restart a new game or exit the app.
- We create an PlayDialog object and pass it to the set...Button methods of AlertDialog.Builder.

# Implementing DialogInterface.OnClickListener (1 of 2)

```
private class PlayDialog implements
                    DialogInterface.OnClickListener {
  public void onClick( DialogInterface dialog,
                                  int id ) {
    // start a new game or exit the app
  }
}
```

# Implementing DialogInterface.OnClickListener (2 of 2)

- The id parameter identifies which button was clicked.
- -1 ➔ Positive (YES) button ➔ start a new game
- -2 ➔ Negative (NO) button ➔ exit the app

# id Is –1: Start a New Game

- Call resetGame of the TicTacToe class to reset the Model side of the game:

  tttGame.resetGame( );
- Enable the Buttons and reset them (no text): add the resetButtons method.
- Reset the TextView to its original look and text.

# resetButtons

```
public void resetButtons( ) {
    for( int row = 0; row < TicTacToe.SIDE; row++ )
      for( int col = 0; col < TicTacToe.SIDE; col++ )
        buttons[row][col].setText( "" );
}
```

# id Is –1: Start a New Game

```
public void onClick( DialogInterface dialog, int id ) {
    if( id == -1 ) /* YES button */ {
        tttGame.resetGame( );
        enableButtons( true );
        resetButtons( );
        status.setBackgroundColor( Color.GREEN );
        status.setText( tttGame.result( ) );
    }
    …
```

# id Is –2: Exit the App (1 of 3)

- We need to call the finish method of Activity ...
- ... but we are in the PlayDialog class.
- To access "this" object of MainActivity from an inner class, we use:

  MainActivity.this

# id Is –2: Exit the App (2 of 3)

- To exit the app:

  ..

  else if( id == -2 )

  MainActivity.this.finish( );

# id Is –2: Exit the App (3 of 3)

```
public void onClick( DialogInterface dialog, int id ) {
    if( id == -1 ) /* YES button */ {
        …
    }
    else if( id == -2 ) // NO button
        MainActivity.this.finish( );
}
```

# Setting the Buttons (1 of 2)

```
public AlertDialog.Builder setPositiveButton(
    CharSequence text,
    DialogInterface.OnClickListener listener )
```

- PlayDialog is a DialogInterface.OnClickListener

# Setting the Buttons (2 of 2)

- Inside showNewGameDialog:

```
// alert is an AlertDialog.Builder
PlayDialog playAgain = new PlayDialog( );
alert.setPositiveButton( “YES”, playAgain );
// similar code for negative button
```

# showNewGameDialog

```
public void showNewGameDialog( ) {
    AlertDialog.Builder alert = new
            AlertDialog.Builder( this );
    alert.setTitle( "This is fun" );
    alert.setMessage( "Play again?" );
    PlayDialog playAgain = new PlayDialog( );
    alert.setPositiveButton( "YES", playAgain );
    alert.setNegativeButton( "NO", playAgain );
    alert.show( );
}
```

# update Method

```
public void update( int row, int col ) {
   …
   if( tttGame.isGameOver( ) ) {
      status.setBackgroundColor( Color.RED );
      enableButtons( false );
      status.setText( tttGame.result( ) );
       showNewGameDialog( );  // play again?
   }
}
```

# Version 5

- In Version 5, we split the View and the Controller (i.e., we use two classes instead of one).
- The objective is to make the View reusable (in addition to the Model).

# View (1 of 2)

- We name the View class GridButtonAndTextView. The whole GUI fits inside a GridLayout, so we make GridButtonAndTextView a subclass of GridLayout.
- It should be reusable and therefore independent from the Model.

# View (2 of 2)

```
public class ButtonGridAndTextView extends
        GridLayout {
    private int side;
    private Button [][] buttons;
    private TextView status;
```

# GridButtonAndTextView (1 of 6)

- In addition to creating the GUI, we should also provide methods to:
  - Update the View
  - Get user input
- Those methods can be called from the Controller (which now has a View instance variable in addition to the Model instance variable).

# GridButtonAndTextView (2 of 6)

- We provide methods to update the View:
  - Set the text of each button.
  - Set the background color of the TextView.
  - Set the text of the TextView.
  - Reset the text of the buttons to the empty String.
  - Enable or disable the buttons.

# GridButtonAndTextView (3 of 6)

```
public void setStatusText( String text ) {
    status.setText( text );
}
public void setStatusBackgroundColor( int color ) {
    status.setBackgroundColor( color );
}
public void setButtonText( int row, int column,
                                        String text ) {
    buttons[row][column].setText( text );
}
```

# GridButtonAndTextView (4 of 6)

```
public void resetButtons( ) {
    for( int row = 0; row < side; row++ )
      for( int col = 0; col < side; col++ )
        buttons[row][col].setText( "" );
}
public void enableButtons( boolean enabled ) {
    for( int row = 0; row < side; row++ )
      for( int col = 0; col < side; col++ )
        buttons[row][col].setEnabled( enabled );
}
```

# GridButtonAndTextView (5 of 6)

- We provide one method to retrieve user input.

  public boolean isButton( Button b, int row,

  int column )

- It returns true if b is the Button at indexes row and column within the array buttons.

# GridButtonAndTextView (6 of 6)

- The constructor accepts four parameters.

```
public ButtonGridAndTextView( Context
    context, int width, int newSide,
    View.OnClickListener listener )
```

# GridButtonAndTextView Constructor (1 of 3)

- Context parameter: we need it to instantiate the Buttons and the TextView.
- int width parameter: that is the width of this GridButtonAndTextView (so that it can be different from the screen's width if we want to).

# GridButtonAndTextView Constructor (2 of 3)

- int newSize parameter: that is the number of buttons per row and column.
- We want to be able to reuse this class with other Models, not just the TicTacToe class, so we include an instance variable for the number of buttons per row and column (i.e., we do not use inside this class the SIDE constant of the TicTacToe class).

# GridButtonAndTextView Constructor (3 of 3)

- View.OnClickListener parameter: we use this to set up event handling. The Buttons are in this class; we register the View.OnClickListener parameter on all the Buttons.
- See Example 3.10.

# GridButtonAndTextView Constructor (Example 3.10)

```
public ButtonGridAndTextView( Context context, int width,
        int newSide, View.OnClickListener listener ) {
    super( context );
    side = newSide;
    // Set up this GridLayout
    // Create the buttons
    /* In order to set up event handling for the Button at
       indexes row and col */
    buttons[row][col].setOnClickListener( listener );
    // create and add TextView at the bottom
}
```

# Controller (1 of 2)

- The MainActivity class is the Controller; it no longer creates the View … but it has an instance variable of type GridButtonAndTextView, a reference to the View.
- With it, it can update the View and and retrieve user input (i.e., identify which button was clicked) by calling methods of GridButtonAndTextView.

# Controller (2 of 2)

```
public class MainActivity extends
                    AppCompatActivity {
   private TicTacToe tttGame;
   private ButtonGridAndTextView tttView;
   …
```

# Controller: Creating the View

```
protected void onCreate( Bundle savedInstanceState ) {
    super.onCreate( savedInstanceState );
    tttGame = new TicTacToe( );
    Point size = new Point( );
    getWindowManager().getDefaultDisplay( ).getSize( size );
    int w = size.x / TicTacToe.SIDE;
    ButtonHandler bh = new ButtonHandler( );
    tttView = new ButtonGridAndTextView( this, w,
        TicTacToe.SIDE, bh );
    tttView.setStatusText( tttGame.result( ) );
    setContentView( tttView );
  }
```

# Dialog Inner Class (1 of 2)

```
private class PlayDialog implements
                                    DialogInterface.OnClickListener {
   public void onClick( DialogInterface dialog, int id ) {
      if( id == -1 ) /* YES button */ {
         tttGame.resetGame( );
         // call methods of the View with tttView
         // in order to prepare the View for a new game
      }
        else if( id == -2 ) // NO button
          MainActivity.this.finish( );
      }
}
```

# Dialog Inner Class (2 of 2)

```
private class PlayDialog implements
                                    DialogInterface.OnClickListener {
   public void onClick( DialogInterface dialog, int id ) {
      if( id == -1 ) /* YES button */ {
         tttGame.resetGame( );
         tttView.enableButtons( true );
         tttView.resetButtons( );
         tttView.setStatusBackgroundColor( Color.GREEN );
         tttView.setStatusText( tttGame.result( ) );
       }
       else if( id == -2 ) // NO button
          MainActivity.this.finish( );
    }
}
```

# Event Handler Inner Class (1 of 3)

```
private class ButtonHandler implements
                    View.OnClickListener {
    public void onClick( View v ) {
      // play
      // call the View methods with tttView to update the View
      // check if the game is over
    }
}
```

# Event Handler Inner Class (2 of 3)

```
public void onClick( View v ) {
        …
   if( tttView.isButton( ( Button ) v, row, column ) ) {
          int play = tttGame.play( row, column );
          if( play == 1 )
            tttView.setButtonText( row, column, "X" );
          else if( play == 2 )
            tttView.setButtonText( row, column, "O" );
           // check if game is over
   }
}
```

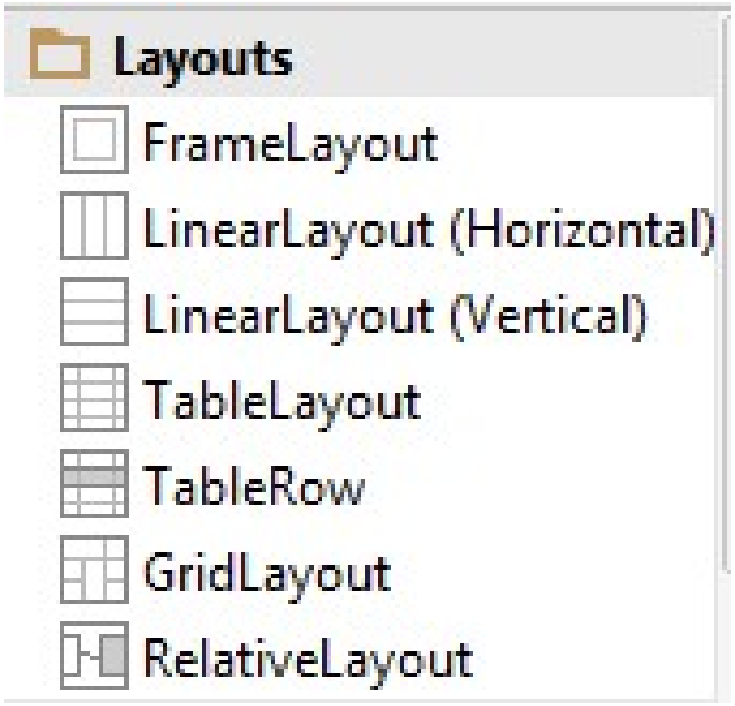
# Event Handler Inner Class (3 of 3)

```
public void onClick( View v ) {
        …
   if( tttView.isButton( ( Button ) v, row, column ) ) {
        …
        if( tttGame.isGameOver( ) ) {
              tttView.setStatusBackgroundColor( Color.RED
);
              tttView.enableButtons( false );
              tttView.setStatusText( tttGame.result( ) );
              showNewGameDialog( );    // offer to play again
         }
   }
}
```

# Model-View-Controller

- In Version 5, the Model (the TicTacToe class) and the View (the GridButtonAndTextView class) are independent of each other and are both reusable.

# Basic Layouts

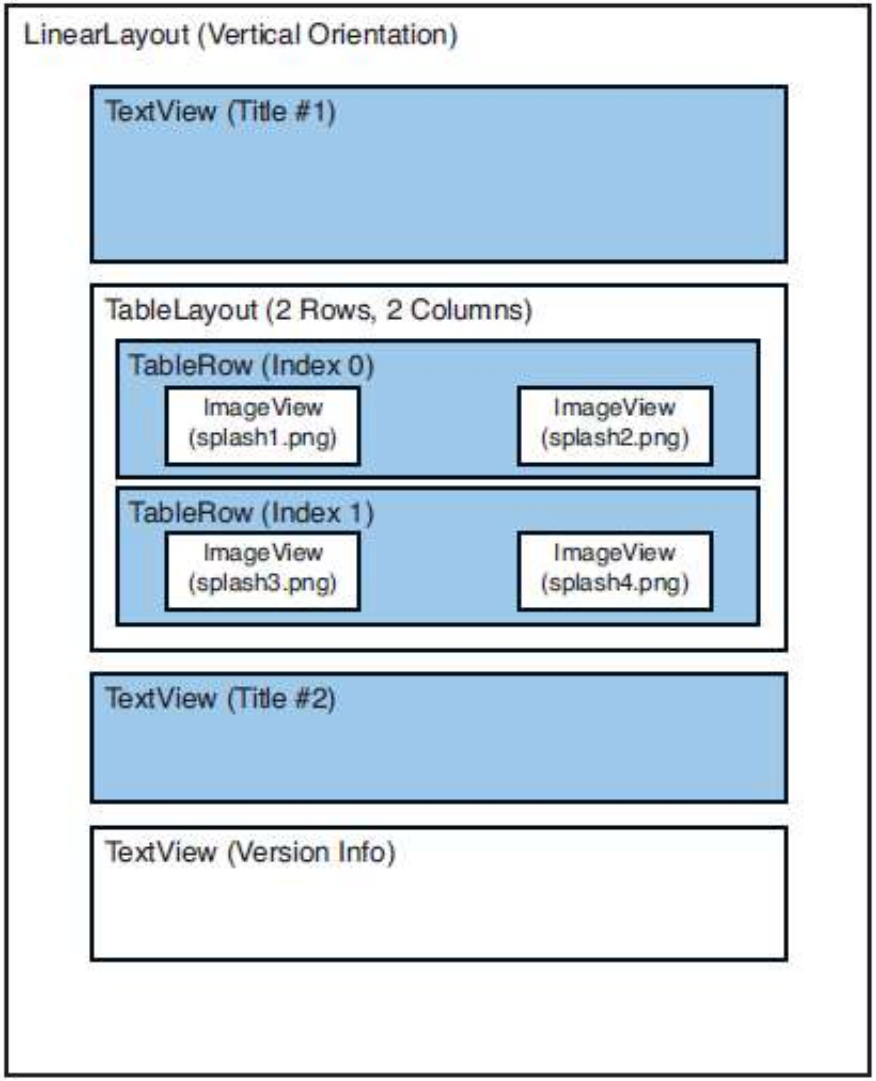


# Basic Layouts

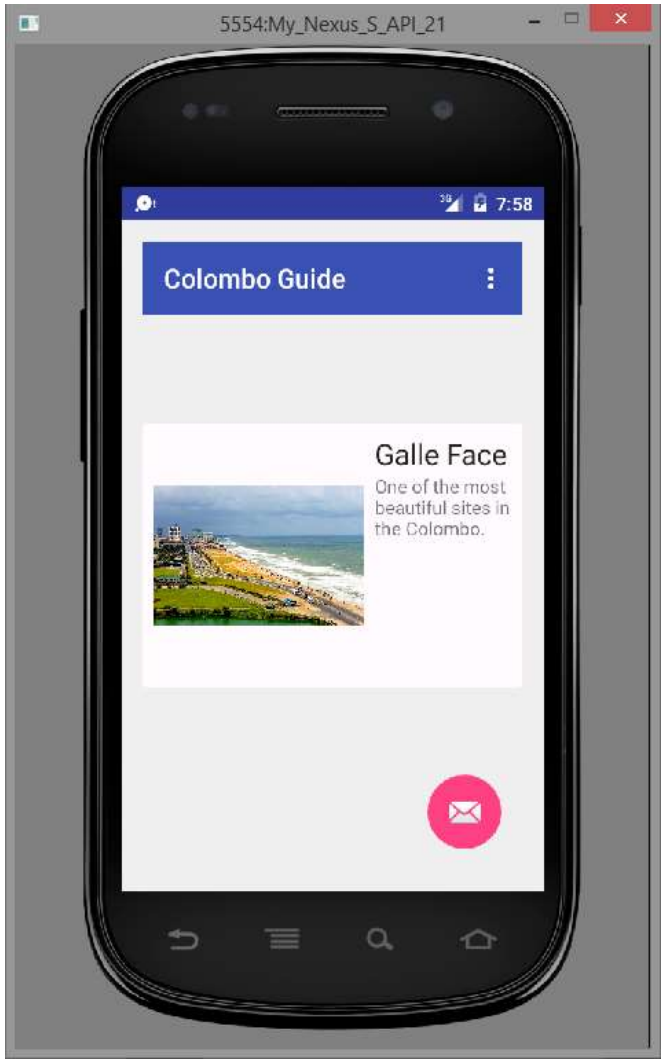
| Layout Control Name | Description | Key Attributes/Elements |
|---|---|---|
| `LinearLayout` | Each child view is placed after the previous one, in a single row or column. | Orientation (vertical or horizontal). |
| `RelativeLayout` | Each child view is placed in relation to the other views in the layout, or relative to the edges of the parent layout. | Many alignment attributes to control where a child view is positioned relative to other child `View` controls. |
| `FrameLayout` | Each child view is stacked within the frame, relative to the top-left corner. `View` controls may overlap. | The order of placement of child `View` controls is important, when used with appropriate gravity settings. |
| `TableLayout` | Each child view is a cell in a grid of rows and columns. | Each row requires a `TableRow` element. |

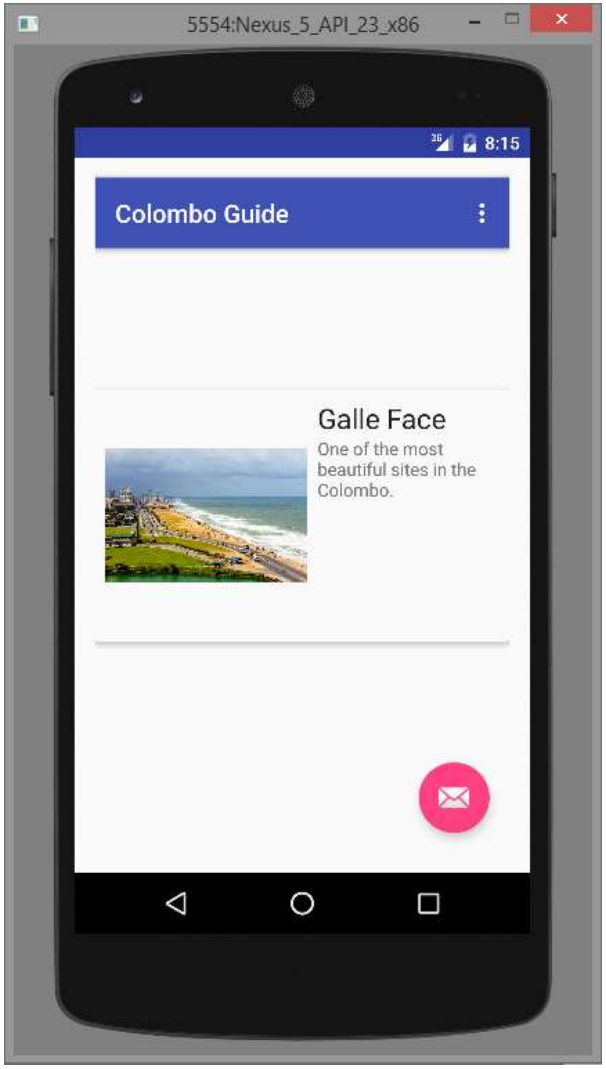
# RelativeLayout

# LinearLayout, TableLayout

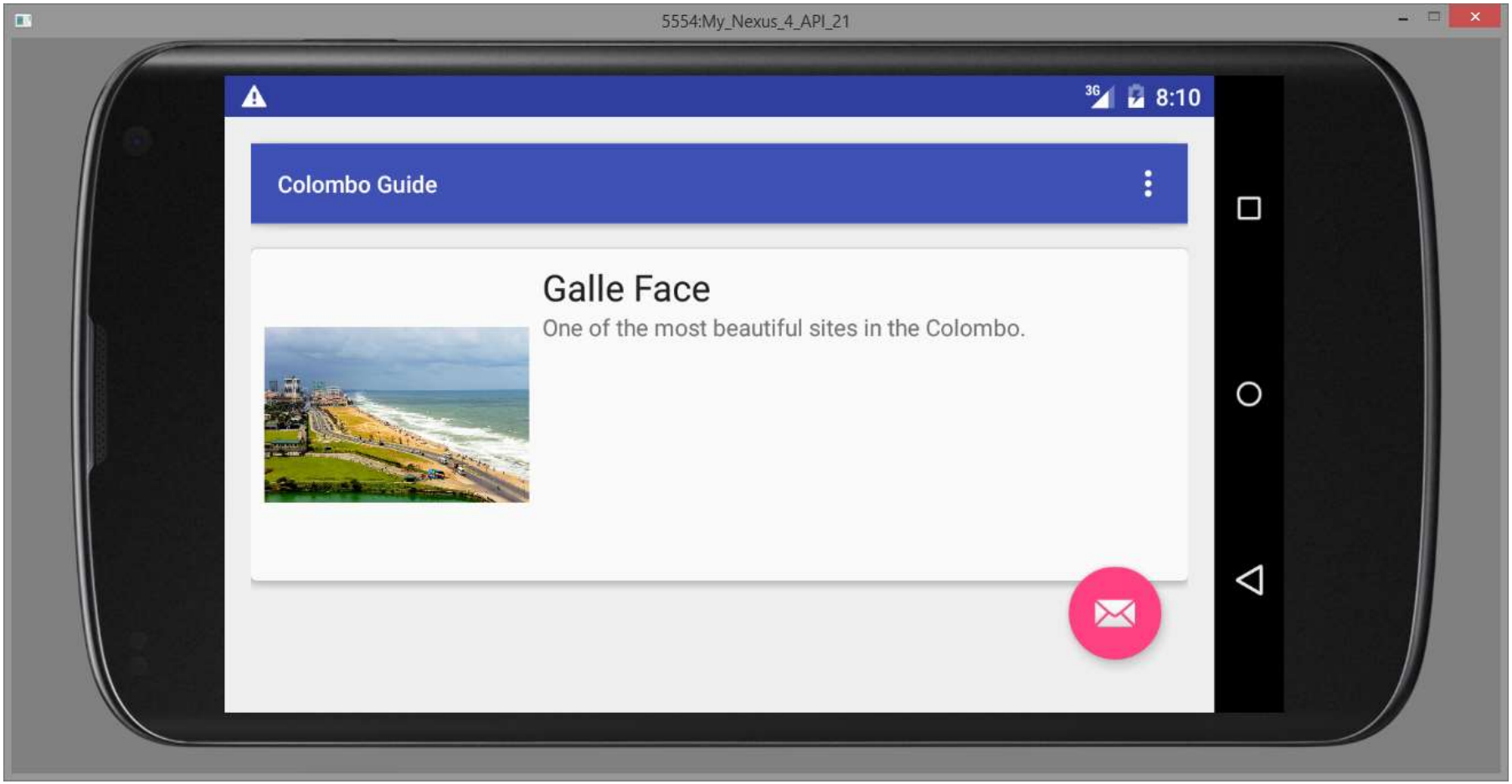


# Example


5554:My_Nexus_S_API_21
7:58
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.


5554:Nexus_5_API_23_x86
8:15
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.

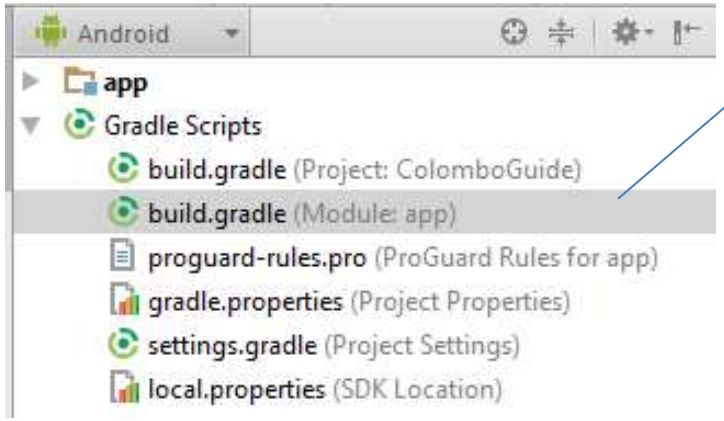
# Example


5554:My_Nexus_4_API_21
8:10
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.

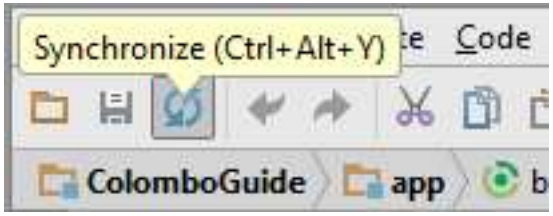
# Adding CardView Layout

1. Open a new Project
2. Delete the *"Hello world"*
3. Update colour and style

## 4. Add CardView library




```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.1.1'
    compile 'com.android.support:cardview-v7:23.1.1'
}
```



# CardView Layout- Updating Resources

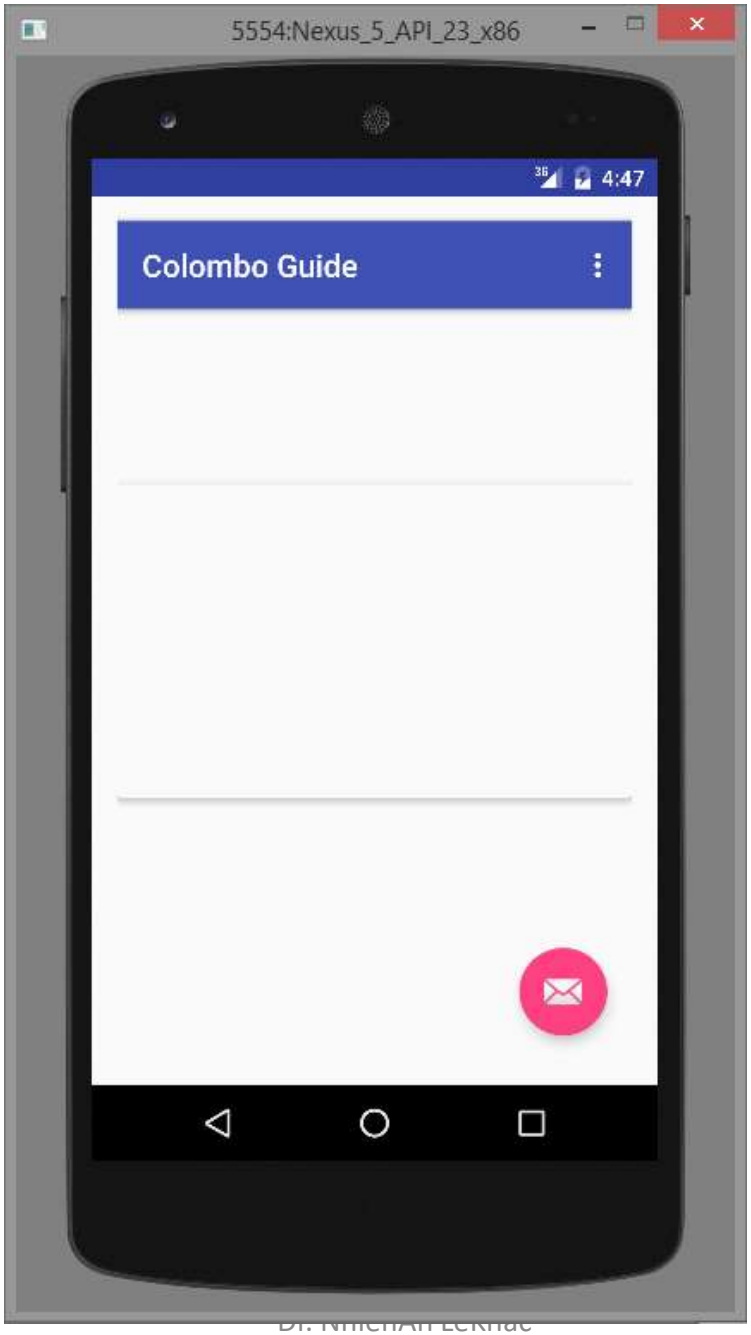
**5. Update *[/res/values/dimens.xml](/res/values/dimens.xml)***

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="fab_margin">16dp</dimen>
    <dimen name="card_height">200dp</dimen>
    <dimen name="card_corner_radius">4dp</dimen>
    <dimen name="card_elevation">3dp</dimen>
    <dimen name="frame_width">160dp</dimen>
    <dimen name="card_padding">8dp</dimen>
</resources>
```

# CardView Layout- Updating Resources

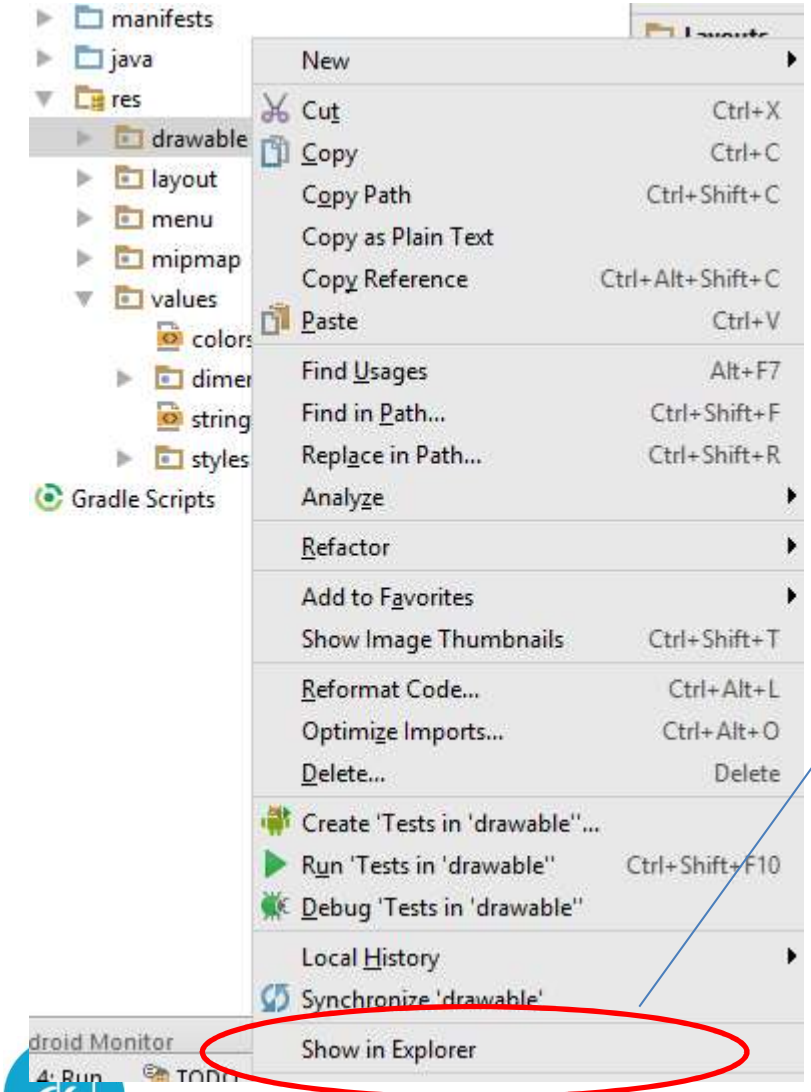
### 6. Update *activity_main.xml*

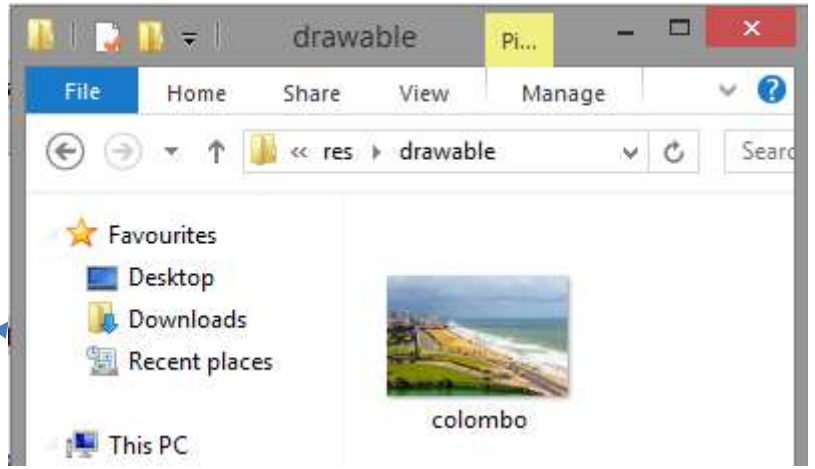
```
<android.support.v7.widget.CardView
    xmlns:card_view=
        "http://schemas.android.com/apk/res-auto"
    android:id="@+id/main_card_view"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_gravity="center"
    card_view:cardCornerRadius="3dp"
    card_view:cardElevation="4dp">
</android.support.v7.widget.CardView>
```


5554:Nexus_5_API_23_x86
4:47
Colombo Guide

# Adding Image

**Open** *res/drawable*



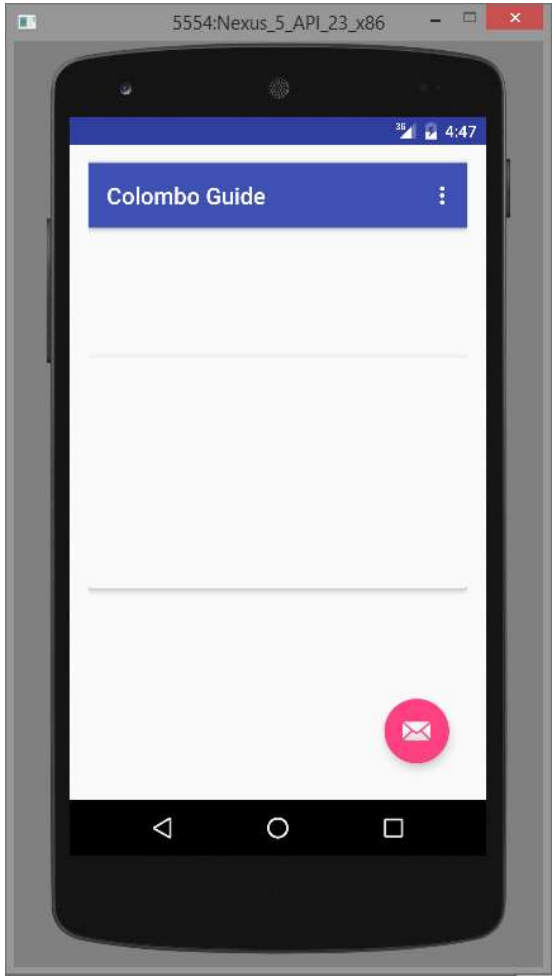



# Adding Text

**Open *res/values/strings.xml***

```
<resources>
    <string name="app_name">Colombo Guide</string>
    <string name="action_settings">Settings</string>
    <string name="title_text">Galle Face</string>
    <string name="detail_text">One of the most beautiful sites in the Colombo.</string>
</resources>
```

# More Resources




1. Relative Layout
2. Frame Layout
3. ImageView
4. TextView

# Update *activity_main.xml*

```
<android.support.v7.widget.CardView
    xmlns:card_view=
        "http://schemas.android.com/apk/res-auto"
    android:id="@+id/main_card_view"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_gravity="center"
    card_view:cardCornerRadius="3dp"
    card_view:cardElevation="4dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="@dimen/card_padding">

        <FrameLayout...>
        <TextView...>
        <TextView...>

    </RelativeLayout>

</android.support.v7.widget.CardView>
```

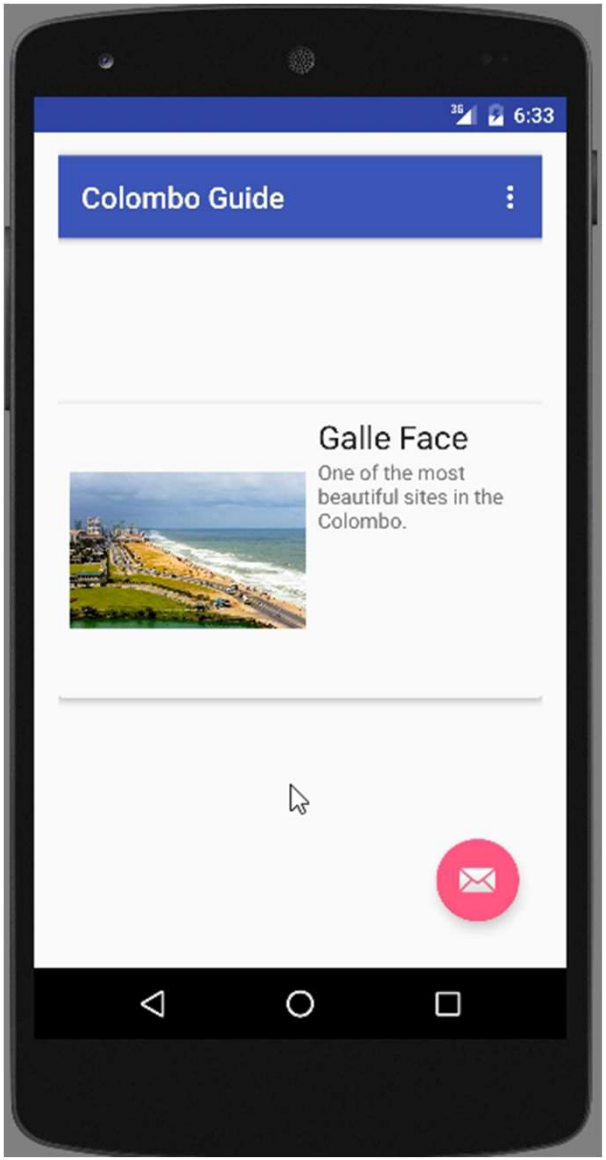
# Update *activity_main.xml*

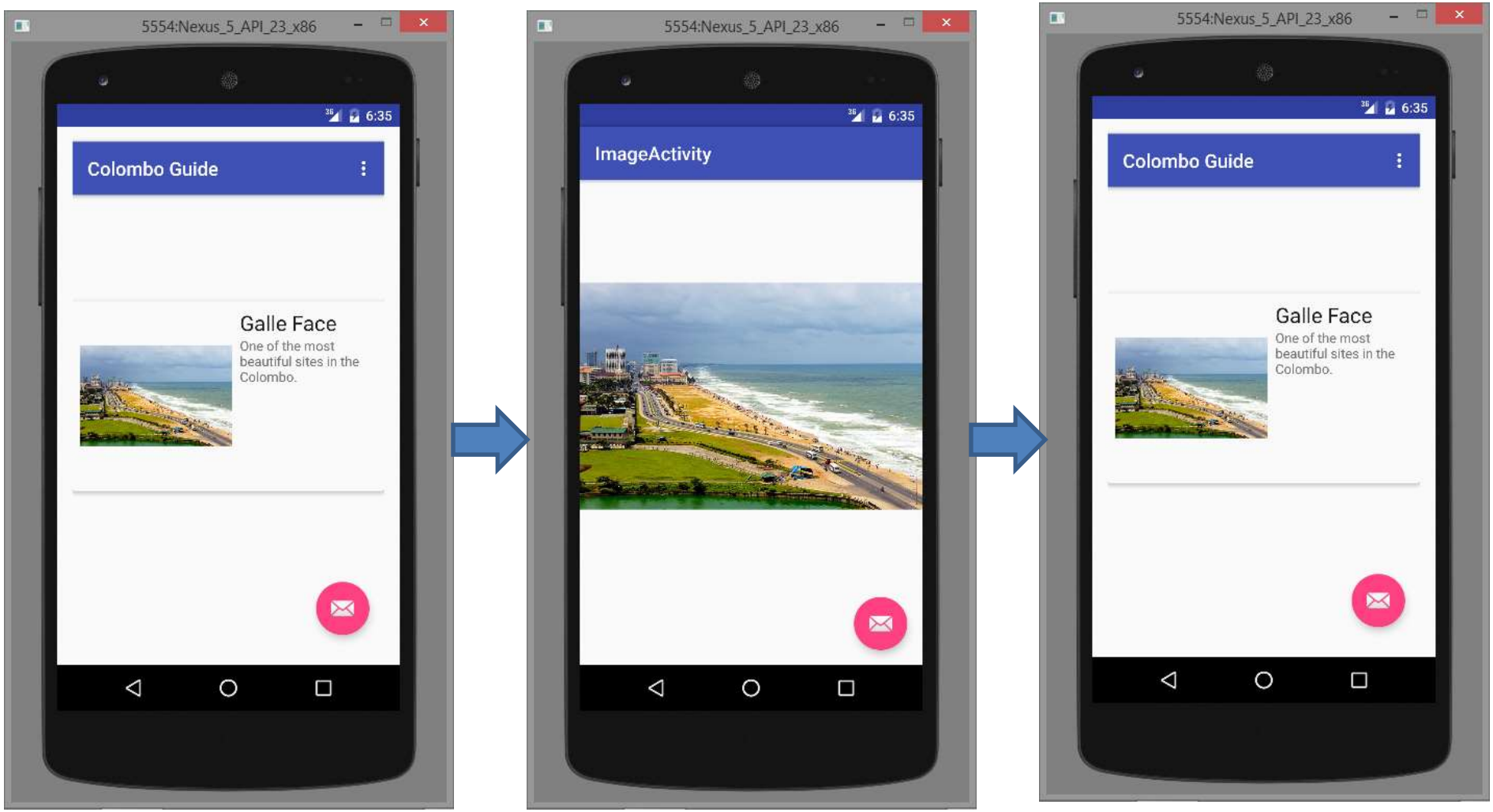
```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/card_padding">

    <FrameLayout
        android:id="@+id/frameLayout"
        android:layout_alignParentStart="true"
        android:layout_centerVertical="true"
        android:layout_height="match_parent"
        android:layout_width="@dimen/frame_width">
        <ImageView
            android:clickable="false"
            android:id="@+id/imageView"
            android:layout_gravity="left|center_vertical"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:src="@drawable/colombo" />
    </FrameLayout>
    <TextView...>
    <TextView...>

</RelativeLayout>
```

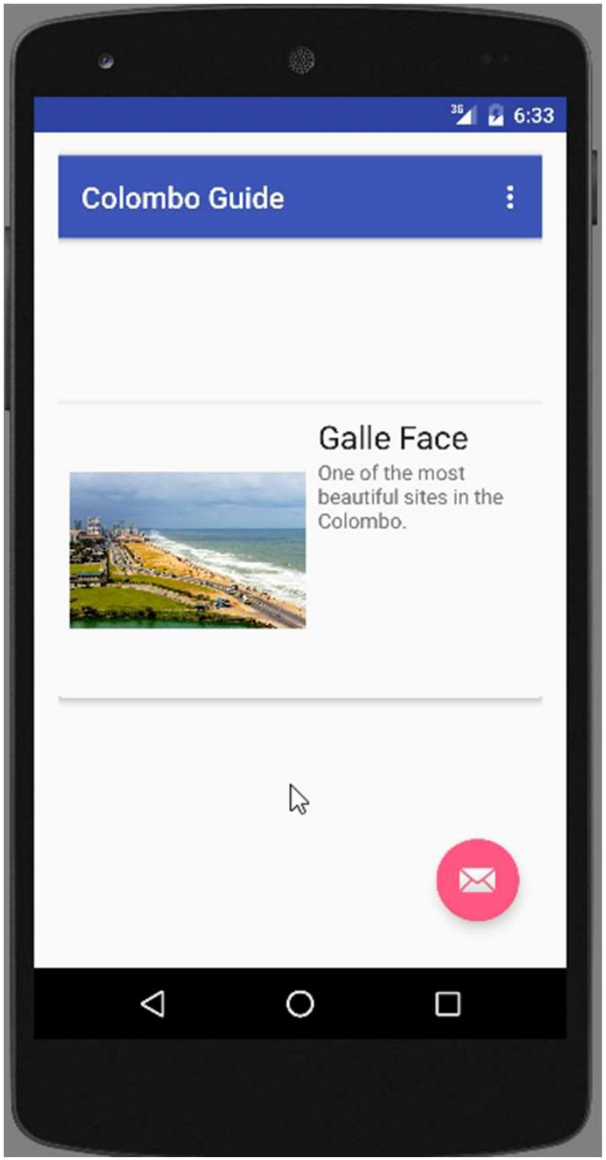
# Update *activity_main.xml*

```
<TextView
    android:id="@+id/title_text_view"
    android:layout_alignParentTop="true"
    android:layout_height="wrap_content"
    android:layout_marginLeft="@dimen/card_padding"
    android:layout_toEndOf="@+id/frameLayout"
    android:layout_width="wrap_content"
    android:text="@string/title_text"
    android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
    android:id="@+id/my_text_view"
    android:layout_below="@+id/title_text_view"
    android:layout_height="wrap_content"
    android:layout_marginLeft="@dimen/card_padding"
    android:layout_toEndOf="@+id/frameLayout"
    android:layout_width="wrap_content"
    android:text="@string/detail_text"
    android:textAppearance="?android:attr/textAppearanceSmall" />
```


# Add new Activity


6:33
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.


5554:Nexus_5_API_23_x86
6:35
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.
5554:Nexus_5_API_23_x86
6:35
ImageActivity
5554:Nexus_5_API_23_x86
6:35
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.


6:33
Colombo Guide
Galle Face
One of the most beautiful sites in the Colombo.