Week 3 Workshop Activity

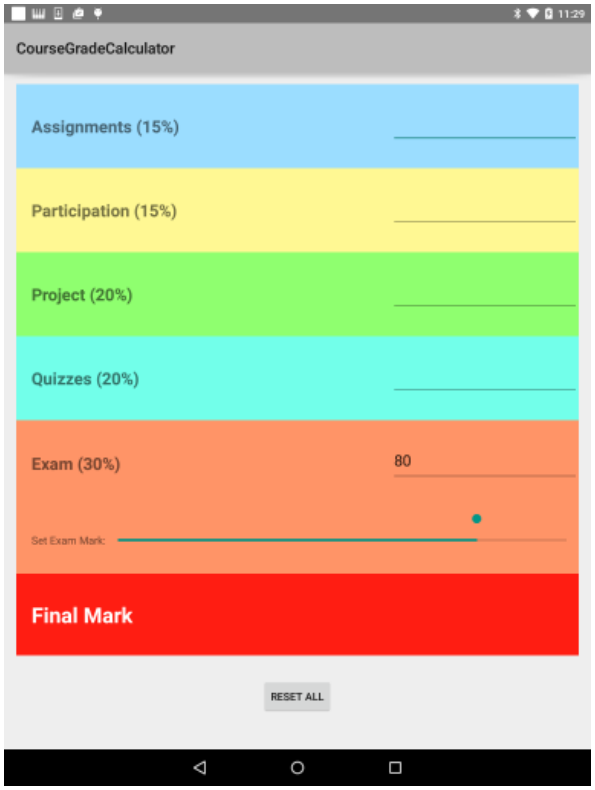
Building the User Interface. Adding Functionality

In this workshop you will build an application that calculates the final mark in a course based on the marks entered by the user for the various components of the course. You will design the GUI and add Java code to specify how the app should process user input and display the results of its calculations.

Objectives of Workshop 3:

1. Design a GUI using LinearLayout and TableLayout and directly edit the XML of the GUI to customize properties.
2. Use UI elements: TextView, EditText, SeekBar, Button.
3. Use event handling to respond to user interactions with the EditText, SeekBar and Button.
4. Programmatically interact with GUI components to change the text they display.

The finalized application will be similar to this:



1. Building the App's GUI

First, you will build the GUI of the 'Course Grades' application. Use a Root LinearLayout (Vertical). Within this top-level vertical LinearLayout, you will have a TableLayout and a HorizontalLinearLayout.

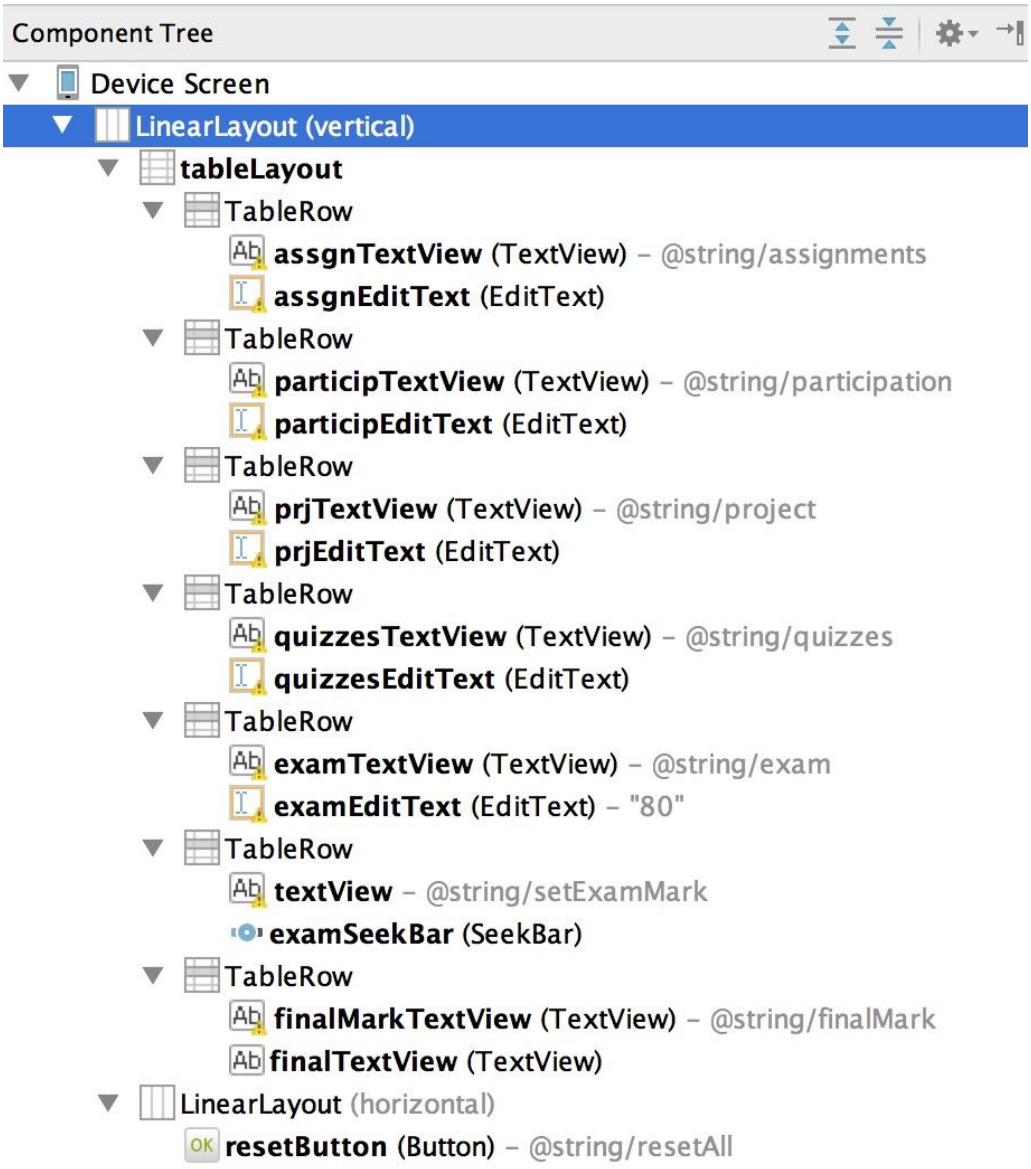
The HorizontalLinearLayout will hold only the 'Reset' button, while the TableLayout will have 7 rows to hold all the other elements. Since the TableLayout holds 7 rows, we can set its weight to 7, and the weight of the HorizontalLinearLayout to 1:

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="7"
    android:id="@+id/tableLayout">

    <TableRow...>

    <TableRow...>

    <TableRow...>

    <TableRow...>

    <TableRow...>

    <TableRow...>

    <TableRow...>

</TableLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:gravity="center">

    <Button...>
</LinearLayout>
</LinearLayout>
```

Add the components for each TableRow, according to the component tree below. Drag each component from the Palette and position it on the proper row – for example drag a TextView from the Palette onto the tableRow1 in the Outline window.

Then change the ID to each component to a meaningful name and also change the labels using the string resources.

Note: it is important to drop these items onto the proper TableRow in the **Outline** window to ensure that the elements are in the right order and nested in the right TableRow object.



2. Adjusting the weight of the components on each TableRow:

First, to have the rows of the table equally distributed, we will assign to each row a weight of 1. Then, in each row the TextView will have a weight of 2 and the EditText will have a weight of 1. This will ensure that all EditTexts are aligned. Exception will make the rows 6 and 7 containing the examSeekbar and final mark display.

TableRow1 is shown below:

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="7"
    android:id="@+id/tableLayout">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#ff9bddff"
        android:gravity="center">

        <TextView
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:text="Assignments (15%)"
            android:id="@+id/assgnTextView"
            android:textStyle="bold"
            android:textSize="22sp"
            android:layout_weight="2"
            android:paddingStart="20dip" />

        <EditText
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:inputType="numberDecimal"
            android:ems="10"
            android:id="@+id/assgnEditText"
            android:layout_weight="1"/>
    </TableRow>
```

Feel free to customize the GUI in any way you wish, however it is important to keep functionality.

TableRow containing the SeekBar:

```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:gravity="center"
    android:background="#ffff9467">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Set Exam Mark:"
        android:id="@+id/textView"
        android:textSize="14sp"
        android:paddingStart="20dip"
        />

    <SeekBar
        android:layout_width="0dip"
        android:layout_height="match_parent"
        android:id="@+id/examSeekBar"
        android:progress="80"
        android:max="100"
        android:layout_weight="2"
        android:indeterminate="false" />

</TableRow>
```

Preventing the user from manipulating text in and examEditText

These controls show calculation results.

Set Focusable, Long Clickable and Clickable properties to False, also CursorVisible in code: *android: CursorVisible = "false".*

3. Adding Functionality to the Application

Class Variables and Instance Variables

The code below declares the instance variables, many of which are the EditTexts into which the user types the various components of the final course grade. The *static Strings* are used as the keys in key/value pairs for the values that will be stored and retrieved in *onSaveInstanceState()* and *onRestoreInstanceState()*, respectively, when the application's configuration changes.

```
private static final String DEBUG_TAG = "CourseGrades";
//constants used when restoring state
private static final String ASSIGNMENTS = "ASSIGNMENTS";
private static final String PARTICIP = "PARTICIP";
private static final String PROJECT = "PROJECT";
private static final String QUIZZES = "QUIZZES";
private static final String EXAM = "EXAM";

private double assgn;  //assignments mark entered by the user
private EditText assgnEditText;

private double particip; //participation mark entered by the user
private EditText participEditText;

private double project; //project mark entered by the user
private EditText projectEditText;

private double quizzes; //quizzes entered by the user
private EditText quizzesEditText;

private TextView finalMarkTextView;  //displays finalMark
```

Once the layout is inflated, get references to the individual widgets using the *findViewById()* method

For each EditText, register the *TextChangedListener* that will respond to events generated when the user changes the text (call the *addTextChangedListener()* method)

Note: your activity will need to implement both View.OnClickListener and TextWatcher.

For the examSeekBar, call the *setOnSeekBarChangeListener()* to register the O*nSeekBarChangeListener* that will respond to events generated when the user moves the thumb of the SeekBar to change the exam mark.

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

    //get references to the controls in the layout
    assgnEditText = (EditText)findViewById(R.id.assgnEditText);
    assgnEditText.addTextChangedListener(this);

    particip EditText = (EditText)findViewById(R.id.particip EditText);
    particip EditText.addTextChangedListener(this);

    projectEditText = (EditText)findViewById(R.id.prjEditText);
    projectEditText.addTextChangedListener(this);

    quizzesEditText = (EditText)findViewById(R.id.quizzesEditText);
    quizzesEditText.addTextChangedListener(this);

    examMarkEditText = (EditText)findViewById(R.id.examEditText);

    finalMarkTextView = (TextView)findViewById(R.id.finalTextView);

    examSeekBar = (SeekBar)findViewById(R.id.examSeekBar);
    examSeekBar.setOnSeekBarChangeListener(examSeekBarListener);

    resetButton = (Button)findViewById(R.id.resetButton);
    resetButton.setOnClickListener(this);

}//end method onCreate()
```

Method *updateStandard()*

This method updates the final mark, each time the user performs some change in any of the EditTexts (full code provided below, as this method does the final mark calculation).

```
//update final mark
private void updateStandard()
{
    // calculate final mark
    double finalMark = assgn*15/100 + particip*15/100 + project*20/100
            + quizzes*20/100 + examVal*30/100;
    //set the text in finalMarkEditText to finalMark
    finalMarkTextView.setText(String.format("%.02f", finalMark));
}
```

Method *updateExam()*

This method has the purpose to update the exam mark in the examMarkEditText based on the value the user selected with the examSeekBar. This value is examVal. Write code for this method: *updateExam().*

```
//update the exam value
private void updateExam()
{
    //set examMarkEditText to match the position of the examSeekBar
    examMarkEditText.setText(String.format("%d",examVal));
}
```

Overriding methods *onSaveInstanceState() and onRestoreInstanceState()* of class MainActivity

We'll store key/value pairs in the *Bundle* that is passed to the *onSaveInstanceState()* method:

```
//save values of assgn, particip, project, quizzes and examVal
@Override
protected void onSaveInstanceState (Bundle outState)
{
    super.onSaveInstanceState(outState);

    outState.putDouble(ASSIGNMENTS, assgn);
    outState.putDouble(PROJECT, project);
    outState.putDouble(QUIZZES, quizzes);
    outState.putDouble(PARTICIP, particip);
    outState.putDouble(EXAM, examVal);
}
```

Write code to retrieve these values using the onRestoreInstanceState() method.

Anonymous inner class that implements interface *OnSeekBarChangeListener*

We create the anonymous inner-class object *examSeekBarListener* that responds to examSeekBar events. The method *onProgressChanged()* is overridden. The method *getProgress()* of SeekBar returns an integer in the range 0-100 representing the position of the SeekBar's thumb. This value is assigned to examVal. We can log this value (for debugging purposes), and we have to call *updateExam()* to update the editText that displays the exam mark and also *updateStandard()* to update the final mark.

```
//called when the user changes the position of the examSeekBar
private OnSeekBarChangeListener examSeekBarListener = new OnSeekBarChangeListener()
{
    //update examVal, then call updateExam(), updateStandard()
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)
    {
        //sets examVal to position of the SeekBar's thumb
        examVal = seekBar.getProgress();
        Log.i(DEBUG_TAG, "examVal"+examVal);
        updateExam();  //update examMarkEditText
        updateStandard();
    } // end method onProgressChanged()

    @Override
    public void onStartTrackingTouch(SeekBar seekBar)
    {
    } //end method onStartTrackingTouch()

    @Override
    public void onStopTrackingTouch(SeekBar seekBar)
    {
    } //end method onStopTrackingTouch
};//end OnSeekBarChangeListener

//event-handling
```

Overriding method *onTextChanged()* of Interface TextWatcher

The *onTextChanged()* method is called whenever the text in an EditText is modified. This method receives four parameters. We only use *CharSequence s* to get the values that the user enters. The text entered by the user is properly converted to double. To check in which component the user has entered text, we will use *isFocused().* Below is shown a portion of this code; write code for all EditTexts in which the user can enter values.

```
@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
    if (assgnEditText.isFocused())
    {
        try
        {
            assgn = Double.parseDouble(s.toString());
        } //end try
        catch (NumberFormatException e)
        {
            assgn = 0.0;  //default if an exception occurs
        } //end catch
    }

    if (particip EditText.isFocused())
    {
        try
        {
            particip = Double.parseDouble(s.toString());
        } //end try
        catch (NumberFormatException e)
        {
            particip = 0.0;  //default if an exception occurs
        } //end catch
    } //end if

    if (projectEditText.isFocused())
    {
        try
        {
```

We have to implement (or provide an emply shell) for all methods of the interface TextWatcher.

```
@Override
public void afterTextChanged(Editable s) {
    // TODO Auto-generated method stub

}

@Override
public void beforeTextChanged(CharSequence s, int start, int count,
        int after) {
    // TODO Auto-generated method stub

}
```

Reset button

Your Reset button should simply clear all values, and reset the SeekBar to default 80.