CP3406 2019 – Week 3 Practical Exercise

Follow through and complete the tasks below – it's based on Chapter 2 from the textbook.

Note: The "Backwards Compatibility check box" is no longer available in the new project wizard for Android Studio. Backwards compatibility is always on now – so any new project you create will be using backwards compatibility by default.

Task 0 – Answer the Discussion board Question

Answer the following question by posting a new thread to the CP3406 LearnJCU Discussion board "Mobile Ecosystem Discussion":

> In your own words, elaborate on how developers might influence the future direction of the mobile ecosystem. (100 words max)

Objective: Construct the Quicksum App

Imagine you are a school teacher or university lecturer. You'd like a quick way to add up exam marks across 100s of exam papers. Here's the Quicksum App use case:

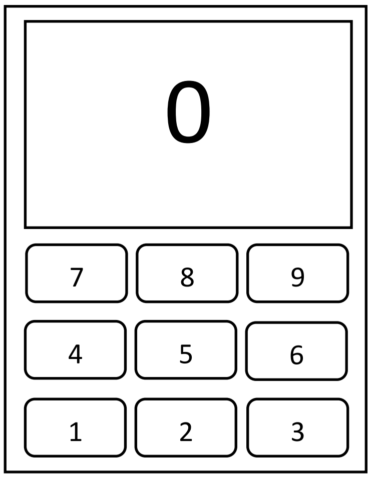
1. After marking an exam paper, open the Quicksum app – it resets the sum to 0
2. Flip through the exam paper and click on the Quicksum app buttons to keep track of marks for each answer
3. When all marks have been entered the App shows the total and can be recorded on the exam paper

The Quicksum App is a cross between a **Utility App** and a **Productivity App**. It has the following characteristics:

- At-a-glance information
- Helps increase the work efficiency of the user
- A minimal and concise UX (e.g. large and clear fonts, easy to read)
- App usage duration is about 10 seconds

The Quicksum App shouldn't be complex. It should only show the most relevant information to the user. It shouldn't present lots of information.

Here is the required UX design for the App:



Task 1 – Construct the UI

1. Create a new empty activity project in Android Studio called “Quicksum”. Set the API level at 19. Be careful to set the folder location: Android Studio remembers the previous folder location 😊

2. In **activity_main.xml**, swap the ConstraintLayout for a vertical LinearLayout, but like this:

```
<android.support.v7.widget.LinearLayoutCompat
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

   So last week we used <LinearLayout>, but not this week. We talk why there are apparently ***two versions of LinearLayout*** later on in class 😊

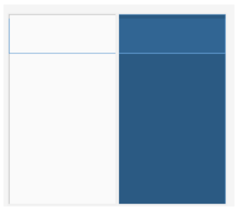
3. Inside the LinearLayout adjust the TextView as follows:

```
<TextView
    android:id="@+id/sum"
    android:textSize="100sp"
    android:layout_marginTop="16dp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

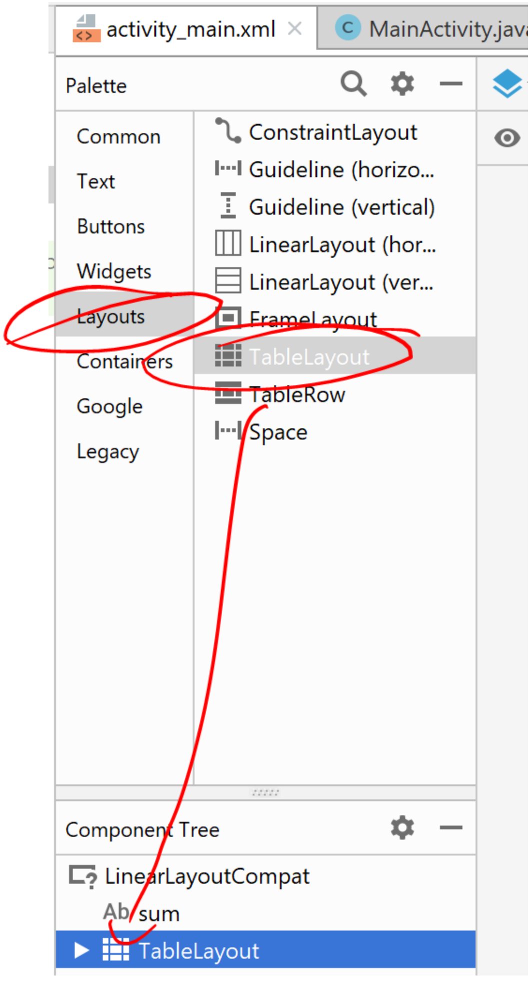
- Recall that the **id** attribute is used to provide access to a UI element programmatically in the corresponding Activity class via (in this case) **R.id.sum**
- dp means “density-independent pixels”
- sp means “scalable pixels” (never use sp for layout_margin or layout_width / layout_height)
- You could use (say) px but that’s absolute pixels. For more check out:

https://developer.android.com/training/multiscreen/screendensities

You should check the WYSIWYG view is something like this:



4. Using the WYSIWYG view, drag a TableLayout down under the TextView:

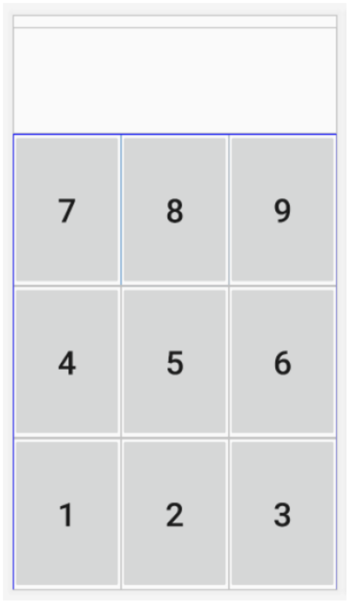


5. Have a look at the XML text editor again, the TableLayout should be populated with a number of TableRow empty tags. Replace them with three TableRows as follows:

```xml
<TableRow
    android:layout_weight="1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <Button
        style="@style/Widget.AppCompat.Button"
        android:textSize="42sp"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />
    <Button
        style="@style/Widget.AppCompat.Button"
        android:textSize="42sp"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />
    <Button
        style="@style/Widget.AppCompat.Button"
        android:textSize="42sp"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />
</TableRow>
```

- The **layout_weight** attributes are useful for telling Android to layout the contents of the row and each of the rows to fill the available space on the screen equally.
- The **style** attribute is useful for setting the “look-and-feel” for UI elements – in this case we are using a predefined Android style (it is possible to make your own…)

6. Add appropriate **text** attributes for each of the buttons to end up with this:



Task 2 – Setup the button handler

1. You might be wondering: “Shouldn’t we also add id attributes to the buttons?”. Well, actually no. Instead let’s make use of the Button tag’s **onClick** attribute – set this attribute the same for all of the buttons. Make it target a method called: “buttonClicked”

2. Add the method signature for buttonClicked() to the **MainActivity.java**

```
public void buttonClicked(View view) {

}
```

- The parameter **view** will actually be the Button that caused the event

3. Add the following “downcast” to buttonClicked():

```
public void buttonClicked(View view) {
    Button button = (Button) view;


}
```

Now you have access to the button that caused the event!

4. Since we know that each button’s text is a number, we can safely convert it to an **int**:
   int number = Integer.parseInt(button.getText().toString());

5. Now, all we need to do is keep track of the sum and update the TextView appropriately!

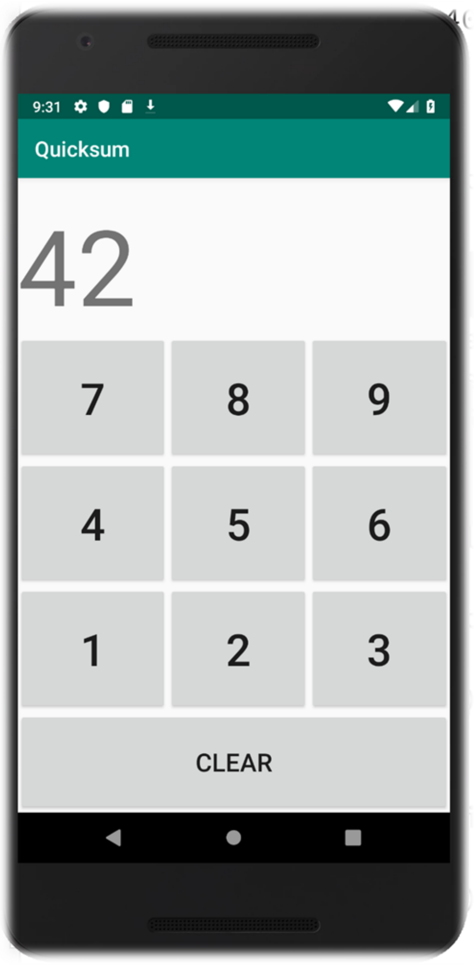
```
public void buttonClicked(View view) {
    Button button = (Button) view;

    int number = Integer.parseInt(button.getText().toString());
    sum += number;

    TextView textView = (TextView) findViewById(R.id.sum);
    String result = "" + sum;
    textView.setText(result);
}
```

- You will need to add a **sum** member field to **MainActivity**

Task 3 – Improve Quicksum

What else could you add to this App to make it more work efficient? What about:

- Go through **activity_main.xml** and **MainAcivity.java** and deal with the “yellow warnings”
- Add a **clear button** that resets the **sum** to 0 (for the TextView and the member field). To do this, you might find it hard to simply add a new Button tag under the TableLayout.
- You might find that you have some duplicate code in **MainAcitivity.java**. Find a way to improve the code – perhaps use a “helper method”?


9:31
Quicksum
42
7
8
9
4
5
6
1
2
3
CLEAR