Android Application Development, COMP 10073

Mohawk College, Winter 2021

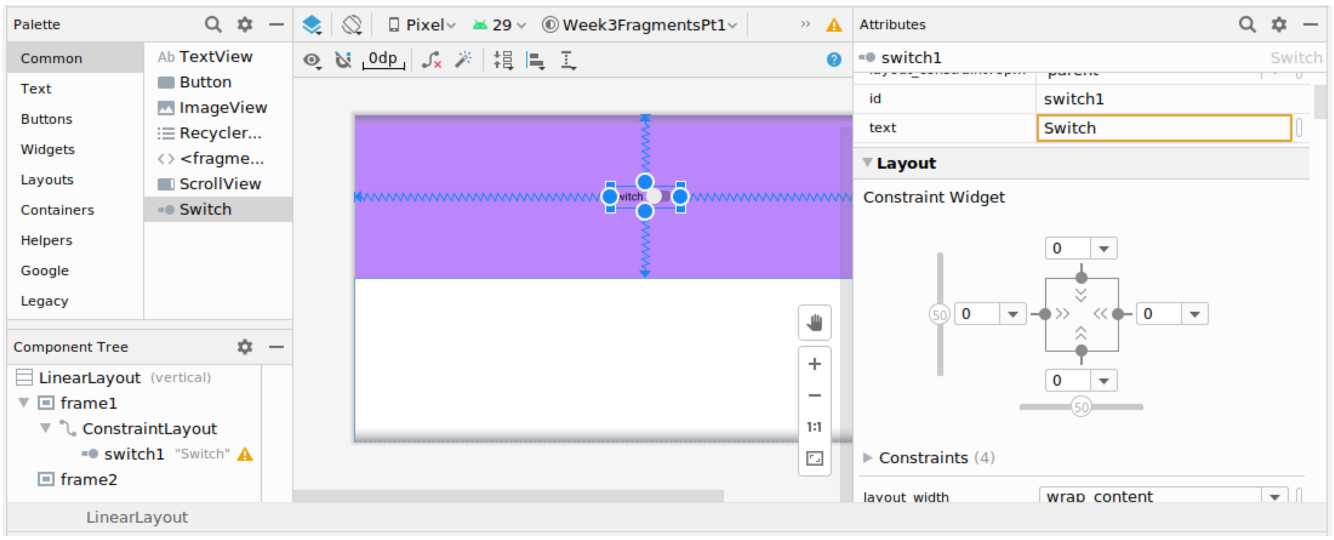
Switch Widget

https://developer.android.com/reference/android/widget/Switch

Homework is graded as pass/fail. Treat the homework instructions as a minimum requirement. You can, of course, add functionality above and beyond the basic requirements presented in the assignments. All the widgets in Android Studio provide variations on behaviour, its good to get practice with as many Android Studio features as you can.

Reconstruct the layout from the previous lecture which filled the UI with two frames. The top frame should use 1/2 of the GUI, and should include a switch. Use a linear layout to balance the frames, add a constrained layout to the top frame so you can centre the switch. Don't add any fragments to activity_main.xml. You can find the switch under the common tab in the widget palette.

Notice that in this layout a fair bit of nesting is used (is this a good thing?).



Component Tree:

```
LinearLayout (vertical)
  frame1
    ConstraintLayout
      switch1 "Switch"
  frame2
```

Attributes (switch1):

id	switch1
text	Switch
layout_width	wrap_content

activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <FrameLayout
        android:id="@+id/frame1"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="@color/purple_200">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <Switch
                android:id="@+id/switch1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:checked="true"
                android:onClick="switchChanged"
                android:text="Switch"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />
        </androidx.constraintlayout.widget.ConstraintLayout>
    </FrameLayout>
    <FrameLayout
        android:id="@+id/frame2"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">
    </FrameLayout>
</LinearLayout>
```

Fragment Manager

https://developer.android.com/guide/fragments/fragmentmanager

https://developer.android.com/guide/fragments/transactions

Instead of adding fragments via XML we can also add fragments programmatically with the fragment manager. FragmentManager is the class responsible for performing actions on your app's fragments.At runtime, a FragmentManager can add, remove, replace, and perform other actions with fragments in response to user interaction. Each set of fragment changes that you commit is called a transaction. You specify what to do inside the transaction using the APIs provided by the FragmentTransaction class. You can group multiple actions into a single transaction—for example, a transaction can add or replace multiple fragments.

Add a Fragment with the FragmentManager

Right click on the file tree to add a fragment to the project. Include a text view widget in the fragment. Don't modify the activity_main.xml, instead we will add the fragment in onCreate().

```
public class MainActivity extends AppCompatActivity {

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

        // Add Fragments by code...  Get Fragment Manager
        FragmentManager fm = getSupportFragmentManager();
        // New Fragment Transaction...
        FragmentTransaction fragmentTransaction = fm.beginTransaction();
        // Create a new instance of our Fragment class
        Fragment myFragment1 = new BlankFragment();
        // Attach the fragment instance to a frame (viewgroup) in our layout.
        // Use .replace to ensure that any previous fragment in the frame is detached.
        fragmentTransaction.replace(R.id.frame2, myFragment1);
        // Commit when done (you can do multiple transactions in a single commit)
        fragmentTransaction.commit();
    }
    . . .
```

The final call on each FragmentTransaction must commit the transaction. The commit() call signals to the FragmentManager that all operations have been added to the transaction.

This looks like more work than just doing it with XML, why bother?

Ultimately we want to develop applications that modify the active fragment based on user decisions. Much like the second lab where we switched activities based on a user choice, in future labs we will swap fragments in and out based on user choices. This example is simply using the fragment manager to replace the second frame with a simple fragment.

Provide a handler for the Switch

The switch class inherits from the button class. It also has an onClick attribute in XML which can be used to specify a handler that modifies the text in the fragment. The switch has a default state, the attribute is named “checked”, set it to true.

This switch handler displays the time that it is changed.

```
public void switchChanged(View view) {

    SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    Date date = new Date();

    Switch sview = (Switch) view;
    TextView textView = findViewById(R.id.textview);
    String switchText = "OFF @ " + formatter.format(date);

    if (sview.isChecked()) {
        switchText = "ON  @ " + formatter.format(date);;
    }
    textView.setText(switchText);
}
```

Application Context

https://developer.android.com/reference/android/content/Context

The Context class provides an interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

The Shared Preferences file is accessed through the application context. This file collects together a bundle of data which can be used to store state information about an application that persists between application restarts.

Save the Switch State with Shared Preferences

https://developer.android.com/reference/android/content/SharedPreferences

Shared Preferences provides an interface for accessing and modifying preference data returned by Context#getSharedPreferences. For any particular set of preferences, there is a single instance of this class that all clients share.

Modifications to the preferences must go through an Editor object to ensure the preference values remain in a consistent state and control when they are committed to storage. Objects that are returned from the various get methods must be treated as immutable by the application.

This class provides strong consistency guarantees. It uses expensive operations which might slow down an app. Frequently changing properties or properties where loss can be tolerated should use other mechanisms. For more details read the comments on Editor#commit() and Editor#apply().

We really should only load our app state when the app starts up, and save the state when it pauses. Fortunately we know something about the activity lifecycle.

Reading Shared Preferences

https://developer.android.com/reference/android/content/Context#getSharedPreferences(java.lang.String,%20int)

The getSharedPreferences method retrieves and hold the contents of the preferences file, returning a SharedPreferences through which you can retrieve and modify its values. Only one instance of the SharedPreferences object is returned to any callers for the same name, meaning they will see each other's edits as soon as they are made.

This method is thread-safe. If the preferences directory does not already exist, it will be created when this method is called.

Fetch the string that stored our saved state from the onResume() state, just before our app becomes active.

```
@Override
public void onResume(){
    super.onResume();

    sharedPreferences = this.getSharedPreferences("Week4", Context.MODE_PRIVATE);
    String switchText = sharedPreferences.getString(SW_STATE, "ON");

    TextView textView = findViewById(R.id.textview);
    textView.setText(switchText);
    Switch sview =  findViewById(R.id.switch1);
    if (switchText.startsWith("ON")) {
        sview.setChecked(true);
    } else {
        sview.setChecked(false);
    }
}
```

Saving Shared Preferences

Save our app state as soon as we go into the onPause() state.

```
@Override
public void onPause() {
    super.onPause();
    TextView textView = findViewById(R.id.textview);
    String switchText = textView.getText().toString();
    // use the shared preferences editor object to modify
    // the values saved for our application.
    SharedPreferences.Editor editor = sharedPreferences.edit();
    // There are several put methods, putString is just one
    editor.putString(SW_STATE, switchText);
    editor.apply();
}
```

Drawable Resources

https://developer.android.com/guide/topics/resources/drawable-resource

https://developer.android.com/guide/topics/graphics/vector-drawable-resources

Android supports a wide array of drawable resources. These can be used as backgrounds for widgets, as icons for buttons, or simply be part of the app display.

To use a bitmap file, copy a file into the res/drawable directory.

There are also XML formats that specify drawable resources.

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"

    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24.0"
    android:viewportHeight="24.0">
    <group
        android:name="rotationGroup"
        android:pivotX="10.0"
        android:pivotY="10.0"
        android:rotation="15.0">
        <path
            android:name="vect"
            android:fillAlpha=".3"
            android:fillColor="#FF000000"
            android:pathData="M15.67,4H14V2h-4v2H8.33C7.6,4 7,4.6
7,5.33V9h4.93L13,7v2h4V5.33C17,4.6 16.4,4 15.67,4z" />
        <path
            android:name="draw"
            android:fillColor="#FF000000"
            android:pathData="M13,12.5h2L11,20v-5.5H9L11.93,9H7v11.67C7,21.4 7.6,22
8.33,22h7.33c0.74,0 1.34,-0.6 1.34,-1.33V9h-4v3.5z" />
    </group>
</vector>
```

Modify Switch to Select an Image

https://developer.android.com/reference/android/widget/ImageView

The method setImageResource, can be used to control the image that appears at run time. Resources point to files in the drawable directory, in this example a .png file and an XML file are both used.

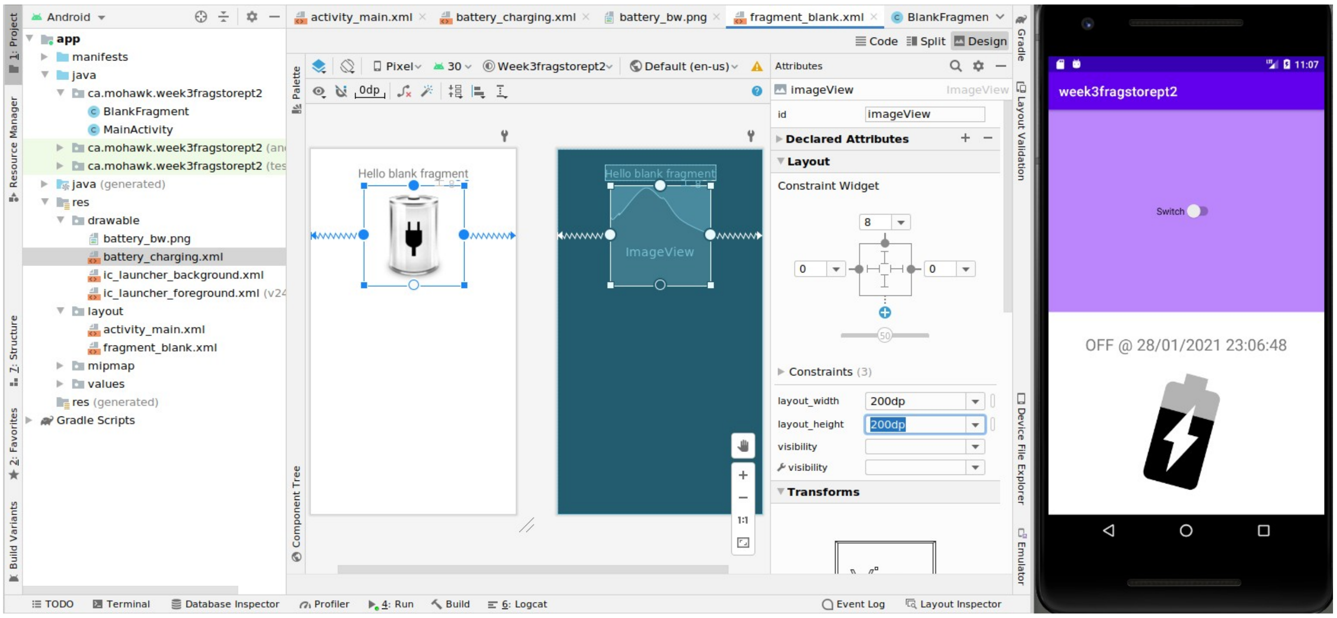
```
public void switchChanged(View view) {

    SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    Date date = new Date();

    Switch sview = (Switch) view;
    TextView textView = findViewById(R.id.textview);
    String switchText = "OFF @ " + formatter.format(date);

    ImageView iview = findViewById(R.id.imageView);

    if (sview.isChecked()) {
        iview.setImageResource(R.drawable.battery_bw);
        switchText = "ON  @ " + formatter.format(date);
    } else {
        iview.setImageResource(R.drawable.battery_charging);
    }
    textView.setText(switchText);
}
```

Android Units

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

https://developer.android.com/guide/practices/screens_support

Android runs on a variety of devices that have different screen sizes and pixel densities. The system performs basic scaling and resizing to adapt your user interface to different screens, but there is more work you should do to ensure your UI gracefully adapts for each type of screen.



id: imageView

layout_width: 200dp

layout_height: 200dp

week3fragstorept2

OFF @ 28/01/2021 23:06:48

Notice that the layout is in “dp” units, aka density-independent pixels.

Convert dp units to pixel units

In some cases, you will need to express dimensions in dp and then convert them to pixels. The conversion of dp units to screen pixels is simple:

px = dp * (dpi / 160)

Other Units

Available units are: px (pixels), dp (density-independent pixels), sp (scaled pixels based on preferred font size), in (inches), and mm (millimeters).