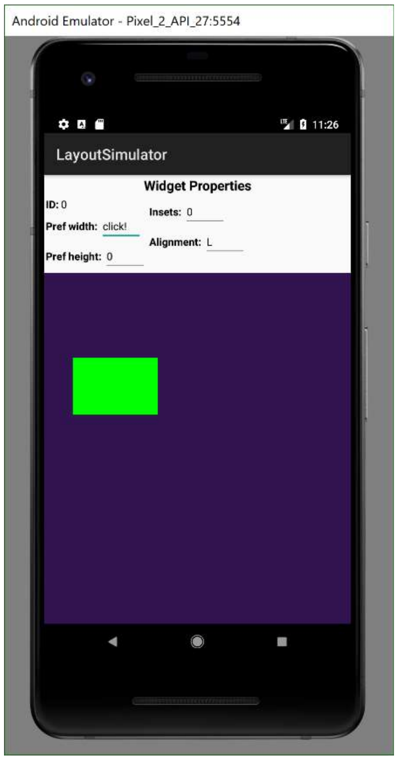
CMPT 381 Assignment 2: Layout

Due: Friday, February 2, 11:55pm

Overview

You will build an interface using several Android widgets and layout containers. You will also build a custom view that does layout for a set of simulated container widgets and simulated buttons. This assignment will demonstrate your ability to work with widgets and layout containers (and their APIs), to use basic event handling, to implement basic elements of layout algorithms, and to implement simple 2D graphical representation of the layout simulation.

Part 1: Android interface with basic layout



Build a simple interface using Android LinearLayouts, TextViews, and Buttons that looks like the picture at right. The interface must include:

- At the top, a text widget with the text "Widget Properties", centred horizontally
- The UI then shows two columns of text labels: in the first column, "ID:", "Pref width:", and "Pref height:"; in the second column, "Insets:" and "Alignment:"
- Beside each label (except "ID:"), there is an editable text field (an EditText); beside "ID:" there is a non-editable text view.
- The rest of the screen is filled with a custom panel that you create (an instance of class LayoutView, as described below). The LayoutView has a purple background and shows a single rectangle in green (left: 100; top: 300; right: 400; bottom: 500).
- When the user touches (or clicks on) the green rectangle, the EditText beside the "Pref width:" label should change to say "click!"
- You should use only LinearLayouts to build the interface
- You can use the Android Studio designer to build (most of) the interface in XML, or you can build the UI programmatically. If you build with XML, you will need to attach variables to the main LinearLayout (to insert the LayoutView), and to the EditText beside "Pref width:" in order to say "click!" after the touch.

The custom View: LayoutView

- Create a custom view that inherits from the View class
- Override the onDraw() method to do your custom drawing
- Add the LayoutView to your interface in your application class
- In your application class, attach an OnTouchListener to the LayoutView to handle touch events.

Resources for part 1:

- Building UIs with Android Studio: https://developer.android.com/training/basics/firstapp/building-ui.html
- Linear Layout tutorial: https://developer.android.com/guide/topics/ui/layout/linear.html
- LinearLayout API: https://developer.android.com/reference/android/widget/LinearLayout.html
- Tutorial for touch listeners: https://developer.android.com/guide/topics/ui/ui-events.html#EventListeners
- Tutorial for custom Views: https://developer.android.com/training/custom-views/create-view.html (note that you should **not** build your LayoutView for the XML editor; it should only be used from the MainActivity code)

Part 2: Adding a layout simulator

In this part of the assignment you will add to the custom LayoutView that you built in part 1. The new version will provide a live graphical simulation of a simple layout manager for a set of simulated linear container widgets and simulated buttons. An example is shown in the figure below (and demonstrations of the working system will be shown in class).

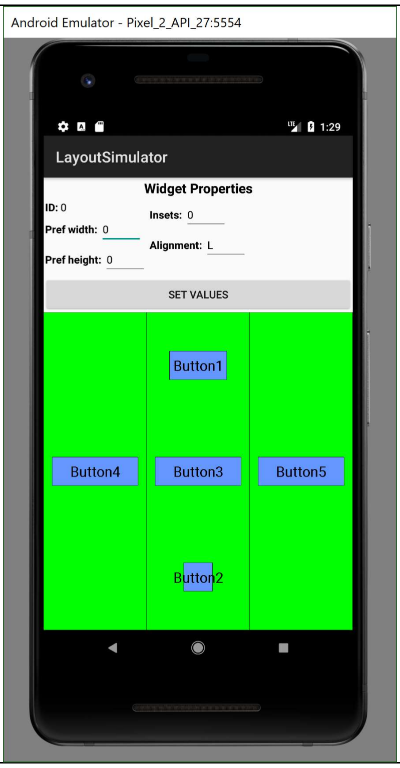
The LayoutView will provide a graphical representation of the layout for a set of custom container and button classes (which you will build). These custom classes are:

- XWidget: the parent class for both the simulated container widget and the simulated button. It can be an abstract class.
- All XWidget objects have an integer id (passed in in the constructor), a preferred width, a preferred height, and an integer insets value (i.e., how much padding will be used around the widget in its allocated area.
- (The alignment value in the interface is unused)
- XFrame: the container class; it has a setting for either HORIZONTAL or VERTICAL layout, similar to a LinearLayout. XFrames maintain a list of their children (which can be XFrames or XButtons). All XFrames are drawn as yellow rectangles.
- XButton: the simulated button class, drawn as a basic blue rectangle with a text label. XButtons have no children.

Requirements for part 2:

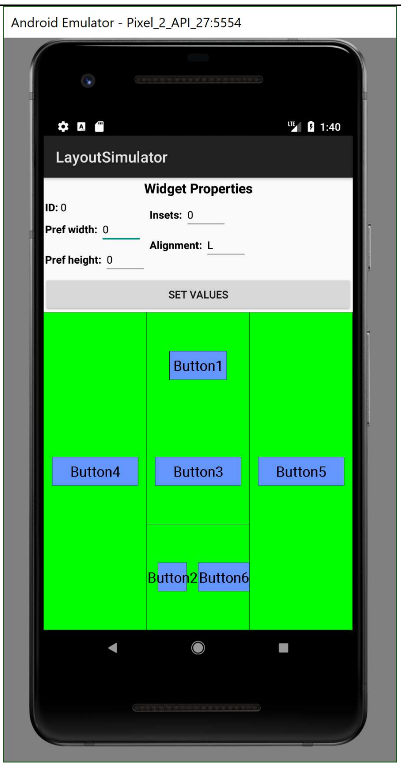
- XFrames always take up all of the available space of their parent, in both dimensions
- XFrames always divide their space up equally among their children (which can be XFrames or XButtons)
- XButtons take up their preferred width and height if it is available.
- If there is not enough space, they shrink to take what is available
- If there is more space available than the preferred size, an XButton stays at its preferred size and aligns to the center of the space allocated to it
- The interface should work appropriately whether the device is in Portrait or Landscape mode
- Clicking on any XButton shows the attributes for that button in the upper part of the UI.
- Once an XButton has been selected, changing the values in the upper UI (and pressing Enter) changes the corresponding values for the button
- Your LayoutView should have a variable of type XWidget that holds the root of the containment hierarchy for your simulated widgets (XFrame and XButton objects)
- Your LayoutView should also create the other XFrame and XButton widgets for your layout simulation, and should set the initial attributes of those objects, and add
- In the onDraw method of your LayoutView, you should call root.layout() with the size of the LayoutView. Each node in your containment tree will set its x, y, width, and height values, and then call layout() on its children, passing in the coordinates of the space available to that child (remember that XFrames allocate space evenly among all children).

```
// Layout code inside LayoutView...

root = new XFrame(1);
root.setOrientation(XFrame.Orientation.HORIZONTAL);

XFrame vert1 = new XFrame(2);
vert1.setOrientation(XFrame.Orientation.VERTICAL);

XButton xb1 = new XButton(3,200,100,"Button1");
xb1.insets = 15;
XButton xb2 = new XButton(4,100,100,"Button2");
xb2.insets = 15;
XButton xb3 = new XButton(5,300,100,"Button3");
xb3.insets = 15;
XButton xb4 = new XButton(6,300,100,"Button4");
xb4.insets = 15;
XButton xb5 = new XButton(7,300,100,"Button5");
xb5.insets = 15;

vert1.addChild(xb1);
vert1.addChild(xb3);
vert1.addChild(xb2);
root.addChild(xb4);
root.addChild(vert1);
root.addChild(xb5);
```



ID: 0
Pref width: 0
Pref height: 0
Insets: 0
Alignment: L

```
// Second example of layout code in LayoutView

root = new XFrame(1);
root.setOrientation(XFrame.Orientation.HORIZONTAL);

XFrame vert1 = new XFrame(2);
vert1.setOrientation(XFrame.Orientation.VERTICAL);

XFrame horiz1 = new XFrame(8);
horiz1.setOrientation(XFrame.Orientation.HORIZONTAL);

XButton xb1 = new XButton(3,200,100,"Button1");
xb1.insets = 15;
XButton xb2 = new XButton(4,100,100,"Button2");
xb2.insets = 15;
XButton xb3 = new XButton(5,300,100,"Button3");
xb3.insets = 15;
XButton xb4 = new XButton(6,300,100,"Button4");
xb4.insets = 15;
XButton xb5 = new XButton(7,300,100,"Button5");
xb5.insets = 15;
XButton xb6 = new XButton(9,300,100,"Button6");
xb5.insets = 15;

vert1.addChild(xb1);
vert1.addChild(xb3);
vert1.addChild(horiz1);
horiz1.addChild(xb2);
horiz1.addChild(xb6);
root.addChild(xb4);
root.addChild(vert1);
root.addChild(xb5);
```



What to hand in

- Java: a zip file of your Android Studio project folders for **either** part 1 (if that is as much as you have completed) **or** part 2. If you have completed part 2, you do not have to hand in a project file for part 1.
- A readme.txt file that indicates exactly what the marker needs to do to run your code. (Systems for 381 should never require the marker to install external libraries).

Where to hand in

Hand in your two files (one zip and one readme.txt) to the link on the course Moodle.

Evaluation

Marks will be given for producing a system that meets the requirements above, and compiles and runs without errors. Note that no late assignments will be allowed, and no extensions will be given, without medical reasons.