* Event-Driven Programming
  + User controls what happens next
    - Most GUIs are event-driven
  + Button press triggers a callback function

Button button = findViewById(R.id.button\_id);

button.setOnClickListener(new View.OnClickListener() {

public void onClick(View v) {

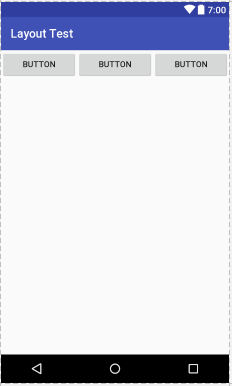
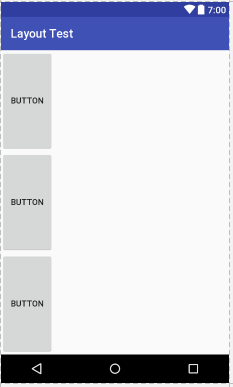
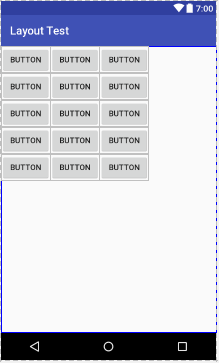
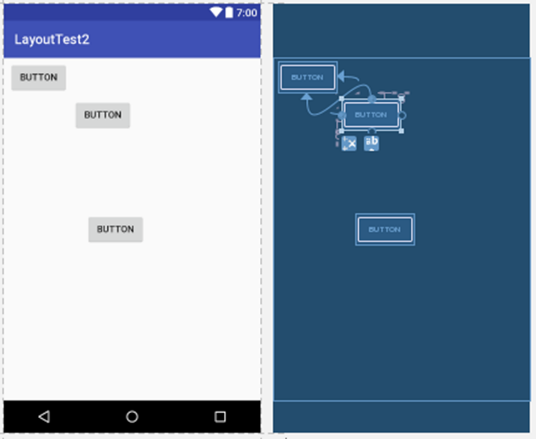
// Code here executes on main thread after user presses button

}

});

* + This will also apply to system events (Activity started, moved to background, etc.)
* Layouts can be created using Java or XML
  + Android Studio layout editor creates XML files
  + Linking an XML file to an Activity:

**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
}

* + Generally the XML approach is easier, but “layouts” that consist of a single graphics area are fine in Java
  + XML allows you to separate the display from the processing
  + May choose to create different layouts for portrait/landscape or phone/tablet
    - Can lock the orientation if the app only makes sense in one orientation
  + Be ready for different screen sizes and aspect ratios, even for phones in portrait orientation
    - 5:3/15:9 (older phones)
    - 16:9 (common)
    - 18.5:9 (newer phones)
    - Other ratios also exist
  + Screens may have rounded corners also
* Cutouts
  + With 9.0 (and some 8.1) the screen may have “notches”
  + Generally only a problem if you hide the navigation/status bar
  + Developer options – Can set to simulate cutout
  + <https://developer.android.com/guide/topics/display-cutout>
* Be ready to test on a variety of devices
* Layout consists of a collection of View and ViewGroup items
* Android Studio includes a layout editor
* Some common layouts
  + LinearLayout (horizontal)
  + LinearLayout (vertical)
  + GridLayout
  + ConstraintLayout
    - Requires use of a compatibility library
    - If no constraints are defined, everything ends up in the top left corner
    - Easy way
      * Drag items from the palette to the layout area
      * Use “Infer Constraints”
    - More work but more control
      * Use the constraint handles on the middle of each side
      * Can add horizontal or vertical guidelines to assist in creating constraints
      * You can right-click to center horizontally or vertically
* View Object
  + Basis of screen elements
  + Defines size, position, color, other basic properties
  + Each view object must have a unique id to be used by the program
    - Static objects don’t need to have an id defined
    - Views in different layouts (portrait/landscape) with the same use will typically be assigned the same id
      * The program can just reference the id and not need to worry about orientation
* TextView
  + Used to display text for the user
    - Could be static text or program output
  + Changing its value:
    - Getting access
      * TextView tv = findViewById( R.id.*tvname* );
      * Note: Older versions of Android Studio required tv = (TextView) findViewById(…)
    - Setting the value
      * tv.setText( “new value” );
      * Can also use values from strings.xml here
* Formatting Text
  + String str = String.format( “format”, values )
  + Format specifiers (lists not complete):
    - %[flags][width][.precision]conversion
    - Flags (optional):
      * - Left Justified
      * 0 Include leading 0s
      * , Include separators
    - Width: Minimum width for the output (optional)
    - Precision: Number of decimal places for a floating-point number (optional)
      * For strings, the maximum number of characters to write
    - Conversions:
      * c Character
      * d Decimal number
      * f Floating-point number
      * % Percent
* EditText
  + Used for user input
  + Includes simple text editing capability, optional auto-complete, cut/copy/paste
  + You may choose to provide a hint (“Name”, etc.)
  + Has an InputType property to indicate what type of input is expected
    - Prevent illegal characters
    - Customizes the soft keyboard
    - List (Table 2-1): *none*, textPersonName, textPassword, numberPassword, phone, textPostalAddress, textMultiline, time, date, number, numberSigned, numberDecimal
  + imeOptions – Customize the action button in the bottom right corner of the keyboard
  + Getting the value:
    - String str = et.getText().toString()
    - Convert to a number, etc., as appropriate
  + Responding to changes:
    - et.setOnFocusChangedListener()
      * Occurs when focus goes to/from the EditText
    - et.addTextChangedListener()
      * Must be an extension of TextWatcher()
      * Implements onTextChanged(), beforeTextChanged(), and afterTextChanged()
* Button
  + User can press it to perform some action
  + btn.setOnClickListener(new View.OnClickListener() {…}
* ToggleButton
  + Switch between two states
  + Use .setOnCheckedChangeListener() to respond when the button is changed
  + Use .isChecked() to check status
  + Use .setChecked() to change status
* Switch
  + Also switches between two states
* Radio Button, CheckBox
  + Only one button in a RadioGroup can be selected
  + Use .isChecked() to check status
  + Use .onClick() to respond to changes
* ProgressBar
  + Indicate progress (such as downloading something)
  + Note difference between ProgressBar and ProgressBar (horizontal)
  + Default progress max is 100, can be changed to what you want in the properties
    - Can set to indicate indefinite progress
  + Change using .setProgress() or .incrementProgressBy()
  + RatingBar – Similar but display in stars
* SeekBar
  + Like a ProgressBar, but user can change the value
  + Get value using .getProgress()
  + Use .setOnSeekBarChangeListener() to respond to changes (includes new progress value)
* Spinner
  + Drop-down list of options
  + Define the list of options in a “string-array” in strings.xml resource file
    - Can also populate it dynamically
  + Two callbacks (part of AdapterView.OnItemSelectedListener)
    - onItemSelected – User selected an item
    - onNothingSelected – Selection disappears from the view
* Number Picker
  + Will have to add using XML
  + Choose from a predefined range of values
  + Set range using .setMinValue() and .setMaxValue()
  + .getValue() and .setValue()
  + Use .setOnValueChangedListener() to respond to changes
* ViewGroup
  + Invisible container for a set of views
* Container Views
  + ListView – vertically scrolling list of items
    - Adapter – Provides the data
    - Must use lv.setAdapter(a) to link the Adapter and the ListView
    - Create a layout for items
  + GridView – Displays items in a two-dimensional grid
  + ExpandableListView – Items can be expanded
  + ScrollView – Vertically scrolling
    - HorizontalScrollView
  + SearchView – Provides a standard way to handle searching
* Design Principles
  + Enchant Me
    - Delight me in surprising ways
      * Subtle effects make it appear easier
    - Real objects are more fun than buttons and menus
      * Less cognitive effort
    - Let me make it mine
      * Allow users to customize the app
    - Get to know me
      * Learn user’s preferences, put defaults and previous choices in easy reach
  + Simplify My Life
    - Keep it brief
    - Pictures are faster than words
    - Decide for me but let me have the final say
      * Set defaults, but allow Undo or modify
    - Only show what I need when I need it
      * Don’t overwhelm the user with too much at once
    - I should always know where I am
      * Provide feedback on where the user is now
    - Never lose my stuff
      * Remember things they took time to create
    - If it looks the same, it should act the same
    - Only interrupt me if it's important
  + Make Me Amazing
    - Give me tricks that work everywhere
      * Leverage patterns and commands from other apps
    - It's not my fault
      * Be gentle about corrections
      * Fix things behind the scenes if possible
    - Sprinkle encouragement
      * Break up complex tasks, give feedback
    - Do the heavy lifting for me
      * Allow novices to do a complex task quickly
    - Make important things fast