Android App Dev 2
Basic UI and Event

CSE 3203
Overview of Mobile Systems

Jack G. Zheng

Fall 2018

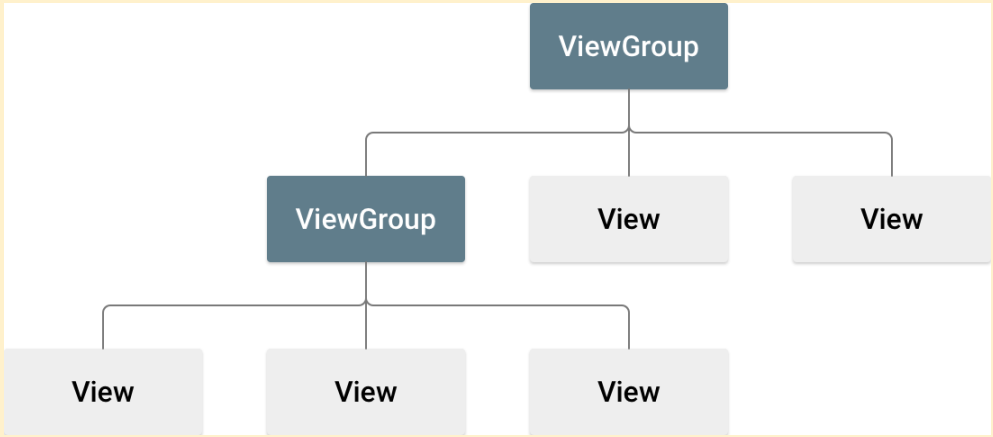
Kennesaw State University

Content Overview

- UI Overview
  - View and ViewGroup
- Layout
  - Declare a layout in XML
  - Static layout options: linear and relative layout
- Basic UI controls and events
  - TextView, EditText, Button
- Simple logic with basic data types

Android App UI Overview

- All user interface elements in an Android app are built using View and ViewGroup objects
  - The View class represents the basic building block for all UI components such as buttons, checkboxes, and text fields.
  - A ViewGroup is an object (an invisible container) that holds other View (and ViewGroup) objects and defines the layout.
  - The user interface is built using a hierarchy of View and ViewGroup objects.



- Android provides a collection of both View and ViewGroup subclasses that offer common input controls (such as buttons, text fields, app bar, and dialogs) and various layout models (such as a linear or relative layout).
- https://developer.android.com/guide/topics/ui/declaring-layout

Layout Building

- The app UI starts with a ViewGroup with a layout
- Two basic ways to build the layout
  - Procedural way: instantiate View objects in Java code and build it at run time. Use this method to dynamically create views based on conditions or iterations (e.g. if or loop).
  - Declarative way: use an XML layout file (as a resource). XML offers a human-readable structure for the layout, similar to HTML. The file is placed as resources in “layout” folder.
- Use the two methods at the same time
  - For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.
- https://developer.android.com/guide/topics/ui/declaring-layout

Advantages using XML

- Separate the presentation of an application from the code that controls its behavior.
- Can modify or adapt UI without having to modify your source code.
- Easier to visualize the structure of the UI, so it's easier to debug problems.

Building XML Layout

- In general, the XML vocabulary (i.e. tag names and attributes) for declaring UI elements closely follows the structure and naming of the corresponding classes and methods in Java.
  - Element (tag) names correspond to class names
  - Attribute names correspond to methods
  - However, note that not all vocabulary is identical. In some cases, there are slight naming differences. For example, the EditText element has a “text” attribute that corresponds to EditText.setText()
- Each layout file must contain exactly one root element, which must be a View or ViewGroup object.
- The structure is a hierarchy tree structure; following XML format standards.
- https://developer.android.com/guide/topics/ui/declaring-layout

Example

LinearLayout is a type of ViewGroup

Layout attributes named layout_*something* define layout parameters for the View/ViewGroup that are appropriate for the ViewGroup in which it resides. This includes layout_width and layout_height

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

id is a common attribute

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

Attributes

- Every View and ViewGroup object supports their own variety of XML attributes
- Some attributes are common to all View objects, because they are inherited from the root View class (like the id attribute)
- Some attributes are specific to a View object
  - for example, TextView supports the textSize attribute
- Some attributes are inherited from its super class

ID

- Any View object may have an integer ID associated with it, to uniquely identify the View within the tree
- When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute. This is an XML attribute common to all View objects (defined by the View class)

```
android:id="@+id/my_button"
```

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file, which holds references to all resources).

Loading a Layout

- Each XML layout file is compiled into a View resource upon compilation.
- A layout file (resource) is loaded in the app code like this:

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

The setContentView() method is used to load the layout file

The reference to the layout file is in the form of: R.layout.layout_file_name. R is the static class for resources.

- The onCreate() callback method is called by the Android framework when your Activity is launched
  - see the discussion about lifecycles: https://developer.android.com/guide/components/activities/activity-lifecycle
- For more information about accessing resources, visit https://developer.android.com/guide/topics/resources/accessing-resources

Create Views

- Define a view in the layout file

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

- Common tasks when defining a view
  - Assign it a unique ID
  - Position and size (defined using layout parameters, and often depending on what layout they are in)

View Position and Size

- The geometry of a view is that of a rectangle.
- The position of the view is expressed as a pair of *left* and *top* coordinates
- The size of the view is two dimensions expressed as a width and a height, and each view is required to define them.
  - layout_height
  - layout_width
- Size values
  - Exact number values: for example, 200dp
  - match_parent: scale to its parent
  - wrap_content: scale just to surround its content
- The unit for location and dimensions is the pixel or dp.
- https://developer.android.com/guide/topics/ui/declaring-layout#SizePaddingMargins

Measuring

- Size and position are usually measured using “dp” or density-independent pixels
  - Density-independent pixels (pronounced “dips”) are flexible units that scale to uniform dimensions on any screen.
  - Use dp to display elements uniformly on screens with different densities, so views are not small on high density screens.
  - dp is only for Android app development
  - https://material.io/guidelines/layout/units-measurements
- A dp is equal to one physical pixel on a screen with a density of 160. To calculate dp:

  dp = (width in pixels * 160) / screen density

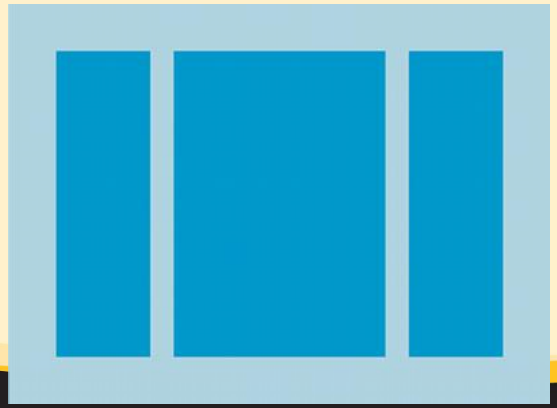
  dp = 160 * width in inches
- A conversion tool
  - https://pixplicity.com/dp-px-converter/

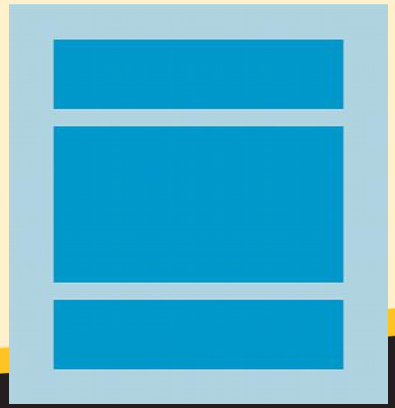
Basic Layout Choices

- Linear Layout
  - A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.
- Relative/Contraint Layout
  - Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).
- https://developer.android.com/guide/topics/ui/declaring-layout#CommonLayouts

Linear Layout

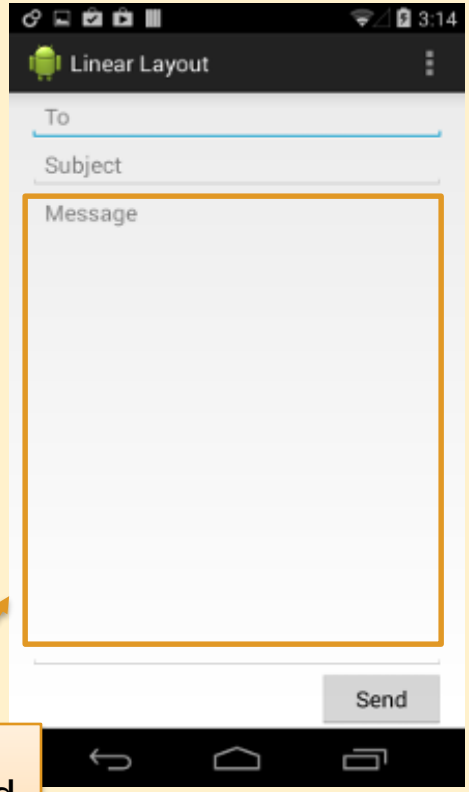
- LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
- You can specify the layout direction with the android:orientation attribute.
- All children of a LinearLayout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding).
- A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.
- https://developer.android.com/guide/topics/ui/layout/linear





Weight

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```



Weight (in linear layouts) can be used to define the priority to auto fill the free space.

Equally Weighted Children

- To create a linear layout in which each child uses the same amount of space on the screen,
  - set the android:layout_height of each view to "0dp" (for a vertical layout) or the android:layout_width of each view to "0dp" (for a horizontal layout).
  - Then set the android:layout_weight of each view to "1".

Relative Layout

- RelativeLayout is a view group that displays child views in relative positions.
- The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).
- A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance.
- If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout
- https://developer.android.com/guide/topics/ui/layout/relative

Relative Layout

- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID). So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on.

- By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from RelativeLayout.LayoutParams

- Some of the layout properties available to views in a RelativeLayout include:
  - android:layout_alignParentTop

  If "true", makes the top edge of this view match the top edge of the parent.
  - android:layout_centerVertical

  If "true", centers this child vertically within its parent.
  - android:layout_below

  Positions the top edge of this view below the view specified with a resource ID.
  - android:layout_toRightOf

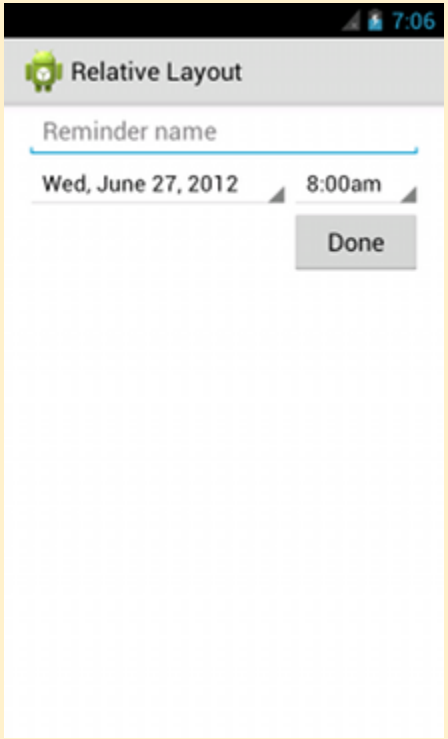
  Positions the left edge of this view to the right of the view specified with a resource ID.

- For more, visit https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams

- Also see https://www.tutorialspoint.com/android/android_relative_layout.htm

Example

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



Comparison

- Relative layout is more flexible to place children views in any position
- Linear layout is more efficient (less settings) when children views are more uniform in size and needs to be aligned well

Constraint Layout

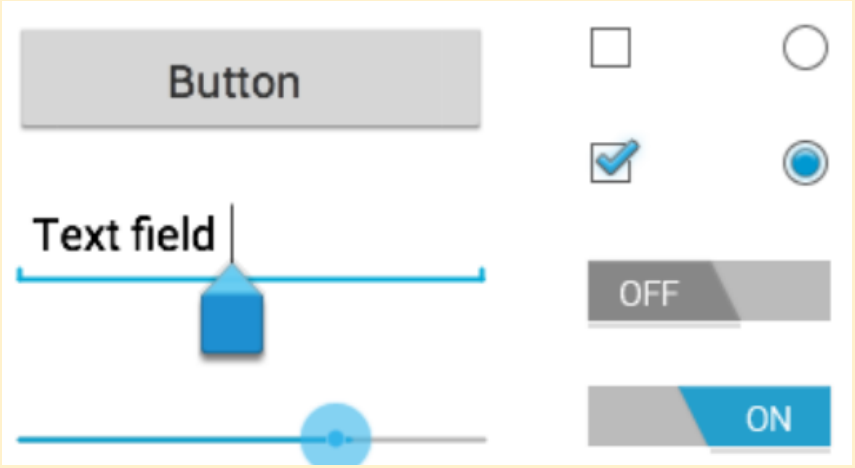
- ConstraintLayout is a more recent choice which allows you to create large and complex layouts with a flat view hierarchy (no nested view groups).
- It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.
- All the power of ConstraintLayout is available directly from the Layout Editor's visual tools, because the layout API and the Layout Editor were specially built for each other. So you can build your layout with ConstraintLayout entirely by drag-and-dropping instead of editing the XML.
- For more information:
  - https://developer.android.com/training/constraint-layout
  - https://constraintlayout.github.io/
  - https://codelabs.developers.google.com/codelabs/constraint-layout/
  - https://medium.com/exploring-android/exploring-the-new-android-constraintlayout-eed37fe8d8f1
  - https://www.youtube.com/watch?v=XamMbnzI5vE
  - https://www.youtube.com/watch?v=z53Ed0ddxgM

Layout Design

- We do not focus on the design aspect
- For some best practices of designing layout in material design, visit
  - https://material.io/guidelines/layout/principles

Common Type of Views

- TextView
- EditText
- Button
- Others
  - ImageButton
  - CheckBox
  - ToggleButton
  - RadioButton
  - RadioGroup
  - Spinner
  - Picker
  - https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-2-user-experience/lesson-4-user-interaction/4-2-c-input-controls/4-2-c-input-controls.html



Reference Views in Code

- Create an instance of the view object and capture it from the layout (typically in the onCreate() method)
  - In XML

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

Reference the view in code by ID

  - In code

```
Button myButton = (Button) findViewById(R.id.my_button);
```

- Then we can manipulate the view (get or set content value, position, size, etc.) by calling its methods
  - getLeft() and getTop(). The former returns the left, or X, coordinate of the rectangle representing the view. The latter returns the top, or Y, coordinate of the rectangle representing the view. These methods both return the location of the view relative to its parent.
  - getWidth(), getHeight()
  - Several convenience methods are offered to avoid unnecessary computations, namely getRight() and getBottom(). These methods return the coordinates of the right and bottom edges of the rectangle representing the view.

TextView

- TextView is the basic control to display text
- Declared in XML layout

```
<TextView
    android:id="@+id/txtTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Kennesaw State University" />
```

- Reference and change text in code

```
TextView textElement = (TextView) findViewById(R.id.txtTitle);
textElement.setText("I love you");
```

- Basic method
  - getText(), setText(), append()
- Reference
  - http://android4beginners.com/2013/06/lesson-1-3-how-to-modify-textview-in-java-code-findviewbyid-settext-and-gettext-methods/
  - https://developer.android.com/reference/android/widget/TextView

Button

- Declare a button in layout XML with click event handler

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

- Define a handler (method) in code

```
/** Called when the user touches the button */
public void sendMessage(View view)
{
    // Do something in response to button click
}
```

- The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must:
  - Be public
  - Return void
  - Define a View as its only parameter (this will be the View that was clicked)

- https://developer.android.com/guide/topics/ui/controls/button

EditText

- This is a UI control for entering and modifying text (a text box)
- Declare an EditText in XML

```
<EditText
    android:id="@+id/plain_text_input"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:inputType="text"/>
```

- Common methods
  - getText(), setText()
- Reference
  - https://developer.android.com/reference/android/widget/EditText

Basic Logic Processing

- All of the following basic processing and coding follow the Java syntax and classes. We will apply them in our examples and assignment, but we do not specifically cover them in lectures. Please refer to any Java resources.
  - Variables, logic control, functions/methods
  - Text processing: string concatenation and styling
  - Number processing: calculation and formatting
  - Dates processing: calculation and formatting
  - Data type conversion

Major Learning Resources

- Main tutorial:
  - https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-1-build-your-first-app/1-2-c-layouts-and-resources-for-the-ui/1-2-c-layouts-and-resources-for-the-ui.html

- Related documentations
  - Layout overview: https://developer.android.com/guide/topics/ui/declaring-layout
    - https://developer.android.com/guide/topics/ui/layout/linear
    - https://developer.android.com/training/constraint-layout/
    - https://developer.android.com/guide/topics/ui/layout/relative
  - Layout editor and inspector:
    - https://developer.android.com/studio/write/layout-editor.html
    - https://developer.android.com/studio/debug/layout-inspector
  - Button: https://developer.android.com/guide/topics/ui/controls/button.html
  - https://developer.android.com/guide/topics/ui/layout/relative
  - Layout editor and inspector:
    - https://developer.android.com/studio/write/layout-editor.html
    - https://developer.android.com/studio/debug/layout-inspector

- Code labs
  - https://codelabs.developers.google.com/codelabs/android-training-layout-editor-part-a/
  - https://codelabs.developers.google.com/codelabs/android-training-layout-editor-part-b/
  - https://codelabs.developers.google.com/codelabs/constraint-layout/

- TutorialsPoint
  - https://www.tutorialspoint.com/android/android_user_interface_layouts.htm
  - https://www.tutorialspoint.com/android/android_user_interface_controls.htm
  - https://www.tutorialspoint.com/android/android_event_handling.htm