CSC 472 / 372 Mobile Application Development for Android

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

Outline

- More on resources
  - Colors
  - Drawables
- Layouts and layout parameters
  - More on *Linear Layout*
- Text input fields

More on Resources

Android Resources – Layout

- Defines the layout of UI (in XML)
- Independent from app logic
  - Separation of concerns, cleaner code
- Allows customization for different screen sizes
  - Different layout, same app logic
- Location: **res/layout/***filename.xml*
- References to layout resources
  - In XML: `@layout/filename`
  - In Java: `R.layout.filename`

Android Resources – String

- Define string constants (in XML)
- Supports localization, e.g., English, French, Chinese
  - Managed by the system
  - You provide the translations of all the phrases in each language and locale, e.g., en_US, en_GB, fr_FR, zh_CN, etc.
- Location: **res/values/strings.xml**
  - Each string has a unique name, e.g., *name*
- Reference to string resources
  - In XML: `@string/name`
  - In Java: `R.string.name`

Color Resources

- Define color constants (in XML)
- Location: **res/values/colors.xml**

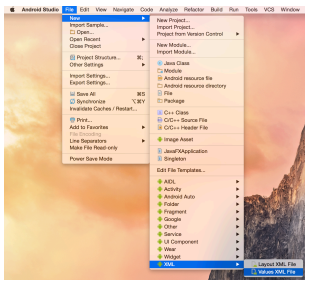
```
<resources>
  <color name="color_name">hex_color</color>
</resources>
```

  - Hex color code in the form of
    #RGB or #RRGGBB or #ARGB or #AARRGGBB
- Reference to color resource
  - In XML: `@color/color_name`
  - In Java: `R.color.color_name`

Add a New XML Values File

- From Android Studio menu bar:

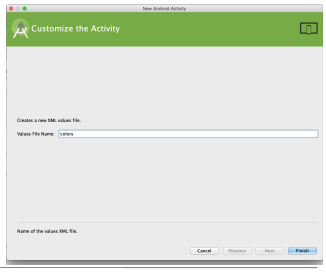
  File | New |
  XML | Values XML File



DePaul University 7

Add a New XML Values File

- Choose a file name (no extension)
- Finish



DePaul University 8

Define Common Colors – colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="bg01">#FF94FFF4</color>
    <color name="red">#FFFF0000</color>
    <color name="red50">#80FF0000</color>
    <color name="blue50">#800000FF</color>
    <color name="green50">#8000FF00</color>
    <color name="yellow50">#80FFFF00</color>
</resources>
```

Opaque colors

50% translucent colors

DePaul University 11

Define Common Colors

The color patches are clickable

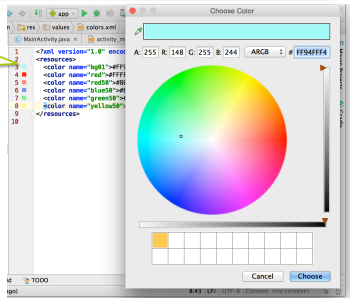
activity_main.xml | colors.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <color name="bg01">#FF94FFF4</color>
4      <color name="red">#FFFF0000</color>
5      <color name="red50">#80FF0000</color>
6      <color name="blue50">#800000FF</color>
7      <color name="green50">#8000FF00</color>
8      <color name="yellow50">#80FFFF00</color>
9  </resources>
10
```

DePaul University 13

Define Common Colors – Using the Color Wheel

Click on the color patch



DePaul University 14

Drawable Resources

- A *drawable* is a general abstraction for "*anything that can be drawn*" in the UI
  - i.e., graphics that can be *drawn* using various techniques
- Graphics rendering is an important factor for app performance and user experience
- Types of drawables include
  - Bitmap images: PNG, JPEG, or GIF
  - Nine patch images: extension of PNG, stretchable
  - XML drawables: shapes, layers, etc.
  - Java objects: implements `Drawable` interface

DePaul University 15

Drawable Resource Folders

- There can be several *drawable* resource folders for different *screen densities* under **res**
- The *default* resource folder
  - **drawable**
- The *alternative* resource folders
  - **drawable-mdpi**
  - **drawable-hdpi**
  - **drawable-xhdpi**
  - **drawable-xxhdpi**

Resources for all densities. Scalable (vector) resources. Bitmaps here will be scaled.

Resources for specific densities. Bitmap resources.

Best Practice: Put bitmaps in these folders.

Provide Drawable Resources

- Each alternative resource folder contains drawable resources for a specific screen density
  - same file name but different sizes/resolutions
  - e.g., add images of different sizes to

res/drawable-mdpi/my_logo.png	(for mdpi devices)
res/drawable-hdpi/my_logo.png	(for hdpi devices)
res/drawable-xhdpi/my_logo.png	(for xhdpi devices)
res/drawable-xxhdpi/my_logo.png	(for xxhdpi devices)

- **Resource optimization**: resources for unused screen densities will be stripped during packaging

Match Drawable Resources to Devices

- Android determines the *best match* of the resources based on the screen density of the device
- If no matching resource is found, it will *scale up* the resource designated for a lower density
  - Fuzzy images. Run-time performance penalty.
- Drawable resources will not be scaled down
  - You must provide all resources for the lowest screen density you intend to support
  - Missing resources will cause apps to crash
    - When the screen density is lower than the minimum density provided and no default drawable resource

Screen Density and Image Size

	DPI	Size	Ratio
ldpi	120	36 x 36	0.75
mdpi	160	48 x 48	1
tvdpi	213	64 x 64	1.33
hdpi	240	72 x 72	1.5
xhdpi	320	96 x 96	2
xxhdpi	480	144 x 144	3
xxxhdpi	640	192 x 192	4

Size for Home screen icons

Reference density 1 dp ≈ 1 px

Reference Drawable Resources

- For image files in **res/drawable*** folders
  - such as *filename.png*
  - The files that contain the same image at different resolutions must have the same name, in their respective folders
- Reference drawable resource
  - In XML:

```
@drawable/filename
```

  - In Java:

```
R.drawable.filename
```

- Resource name is the file name without the extension

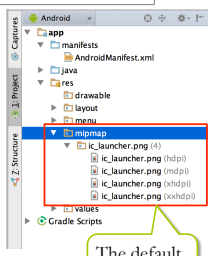
Hello Drawable!

App Icons and Mipmaps

- You should provide app icon images for all screen densities
  - The launcher app picks the best image, avoid up-scaling
- You should define app icons as *mipmaps*
  - A special type of drawable resources for bitmaps
  - A graphics rendering technique that stores pre-rendered results of the same image of different sizes to avoid expensive computations at run-time.
  - Trade-off: memory space for better performance
- No mipmap resource will be stripped during resource optimization (app icons are used by other apps, e.g. *Home*)

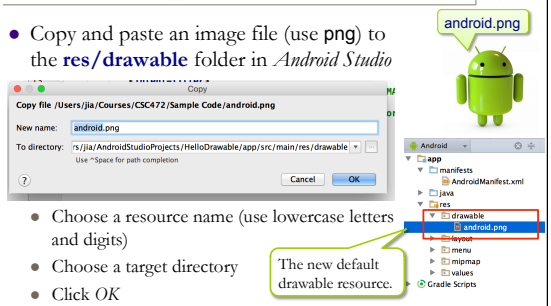
Define App Icon

- Provide a mipmap resource for the app icon
  - Under **res/mipmap**
  - Use *Asset Studio* to automatically generate the images at different resolutions, ahead-of-time (*AOT*)
- Specify the app icon in the *Android Manifest*



Add an Image as Drawable

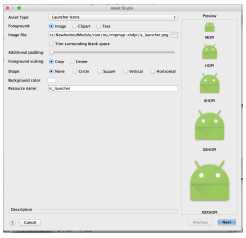
- Copy and paste an image file (use png) to the **res/drawable** folder in *Android Studio*



  - Choose a resource name (use lowercase letters and digits)
  - Choose a target directory
  - Click *OK*

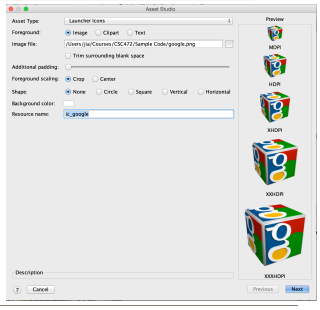
Add an App Icon

- From *Android Studio* toolbar:
  File | New | Image Assets
  - Launch *Asset Studio*



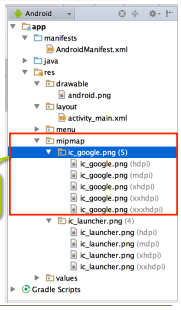
Add an App Icon

- In *Asset Studio*
  - *Asset type*: Launch Icon
  - Select an *Image File*
  - Change the *Resource Name* to ic_google
  - Click *Next*
  - Click *Finish*



Add an App Icon

- *Asset Studio* will automatically generate images for different screen densities
- The images will be added to the **res/mipmap*** folders:
  mipmap-hdpi
  mipmap-mdpi
  mipmap-xhdpi
  mipmap-xxhdpi
  mipmap-xxxhdpi



A Layout with an Image View – activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:paddingLeft="@dimen/activity_horizontal_margin"
              android:paddingRight="@dimen/activity_horizontal_margin"
              android:paddingTop="@dimen/activity_vertical_margin"
              android:paddingBottom="@dimen/activity_vertical_margin"
              tools:context=".MainActivity">
  <TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/android" />
</LinearLayout>
```

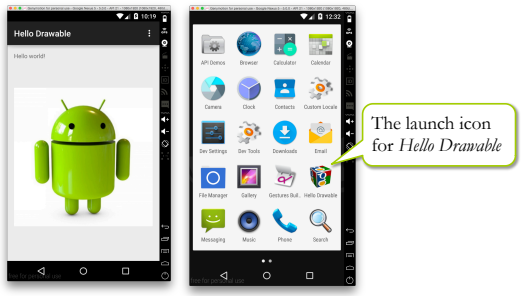
Reference to the drawable resource named *android*

Change App Icon – AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.depaul.csc472.helloandroid" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_google"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

The icon resource

Hello Drawable!



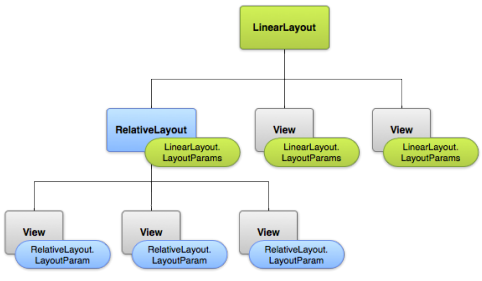
The launch icon for *Hello Drawable*

Managing Layouts

Layout Parameters

- (XML) Attributes of a view object usually are usually concerned with the view object itself
  - e.g., `android:text`, `android:id`
- *Layout parameters* are attributes in the form of `layout_attr`
  - e.g., `android:layout_width`, `android:layout_height`
- Layout parameters of a view object is used by its parent container to determine the layout of the children
- Different layouts support different layout parameters
  - The suitable layout parameters are determined by the parent container.

Layout Parameters



Common Layout Parameters

- Parameters available for all layouts
- Layout *width* & *height*
  `android:layout_width`
  `android:layout_height`
- Layout *margins*: extra space *outside* the view
  `android:layout_margin`
  `android:layout_marginTop`
  `android:layout_marginBottom`
  `android:layout_marginRight`
  `android:layout_marginLeft`

Compare to *padding*: extra space *inside* the view

Linear Layout Parameters

- Parameters specific to the *Linear Layouts*
- *Layout gravity*
  - Determines the positioning and alignment of the view
  - Values: `top` `bottom` `left` `right` `center`
  - More values available. Allow bitwise combination.
- *Layout weight*
  - Determines the amount of extra space is allocated to the view
  - Values are float type. (Default: 0)

Linear Layout Demo

- A simple app to demonstrate the effects of various layout parameters of *Linear Layout*
- The app contains several independent layout files in **res/layout**
  - activity_main_demo1.xml, activity_main_demo2.xml, …
- Change the following line in **MainActivity.java** to load different layout files

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main_demo1);
}
```


Linear Layout Demo – activity_main_demo1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://…"
    android:orientation="vertical"
    android:background="@color/bg01"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView … />
    <TextView … />
    <TextView … />
    <TextView … />
</LinearLayout>
```

Linear Layout Demo – Wrap Content

```
<LinearLayout …>
  <TextView
    android:text="@string/hello_world"
    android:background="@color/red50"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <TextView … />
  <TextView … />
  <TextView … />
</LinearLayout>
```

Linear Layout Demo – Default Layout Gravity & Gravity

```
<LinearLayout …>
  <TextView … />
  <TextView
    android:text="@string/hello_world"
    android:background="@color/yellow50"
    android:layout_width="150dp"
    android:layout_height="50dp" />
  <TextView … />
  <TextView … />
</LinearLayout>
```

Linear Layout Demo – Layout Gravity vs. Gravity

```
<LinearLayout …>
  <TextView … />
  <TextView … />
  <TextView
    android:text="@string/hello_world"
    android:background="@color/green50"
    android:layout_width="150dp"
    android:layout_height="50dp"
    android:layout_gravity="center" />
  <TextView … />
</LinearLayout>
```

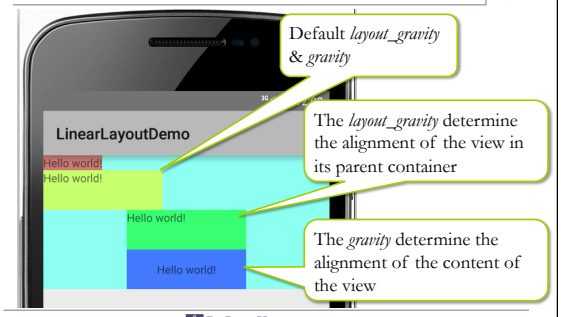
DePaul University 48

Linear Layout Demo – Layout Gravity vs. Gravity

```
<LinearLayout …>
  <TextView … />
  <TextView … />
  <TextView … />
  <TextView
    android:text="@string/hello_world"
    android:background="@color/blue50"
    android:layout_width="150dp"
    android:layout_height="50dp"
    android:layout_gravity="center"
    android:gravity="center" />
</LinearLayout>
```

DePaul University 49

Linear Layout Demo – activity_main_demo1.xml



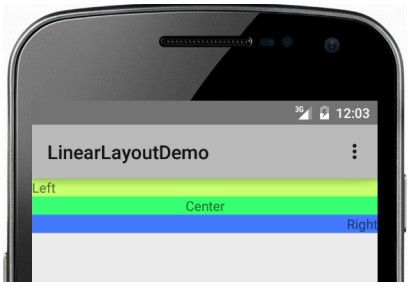
DePaul University 53

Linear Layout Demo – activity_main_demo2.xml

```
<LinearLayout … >
  <TextView android:text="Left"
        android:background="@color/yellow50"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="left" />
  <TextView android:text="Center"
        android:background="@color/green50"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
  <TextView android:text="Right"
        android:background="@color/blue50"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right" />
</LinearLayout>
```

54

Linear Layout Demo – Effect of Gravity



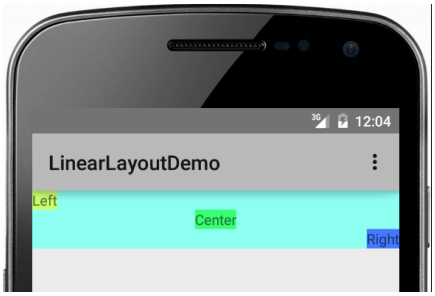
DePaul University 55

Linear Layout Demo – activity_main_demo3.xml

```
<LinearLayout … >
  <TextView android:text="Left"
        android:background="@color/yellow50"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left" />
  <TextView android:text="Center"
        android:background="@color/green50"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />
  <TextView android:text="Right"
        android:background="@color/blue50"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right" />
</LinearLayout>
```

56

Linear Layout Demo – Effect of Layout Gravity



DePaul University 57

Effects of Layout Weight

- *Layout weight* reflects the "importance" of each child in a container
- When there is extra space in a container, the extra space is distributed proportionally based on the `layout_weight` of each child
- When every child has 0 layout weight (default), none gets extra space. All extra spaces are allocated to the end.
- When only one child has layout weight > 0, it gets allocated all the extra space

DePaul University 58

Linear Layout Demo – activity_main_demo4.xml

```
<LinearLayout
  android:orientation="vertical"
  android:background="@color/bg01"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/yellow50"
    android:text="@string/first" />
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/middle04"
    android:textColor="@color/red" />
  …
</LinearLayout>
```

DePaul University 59

Linear Layout Demo – activity_main_demo4.xml

```
<LinearLayout … >
  …
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/yellow50"
    android:text="@string/last" />
</LinearLayout>
```

The container is tightly wrapped. No extra space.

DePaul University 61

Linear Layout Demo – activity_main_demo5.xml

```
<LinearLayout
  android:orientation="vertical"
  android:background="@color/bg01"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <TextView … />
  <TextView … />
  <TextView … />
</LinearLayout>
```

The extra space is allocated at the bottom.

DePaul University 63

Linear Layout Demo – activity_main_demo6.xml

```
<LinearLayout
  android:orientation="vertical"
  android:background="@color/bg01"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <TextView … />
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/middle06"
    android:textColor="@color/red" />
  <TextView … />
</LinearLayout>
```

The extra space is allocated to the middle child.

DePaul University 65

Evenly Divided Space

- To evenly divide the width of a container among all its children
  - Set the `layout_width` of each child to 0 dp
  - Set the `layout_weight` of each child to an equal value
- To evenly divide the height a container among all its children
  - Set the `layout_height` of each child to 0 dp
  - Set the `layout_weight` of each child to an equal value

DePaul University 66

Linear Layout Demo – activity_main_demo7.xml

```
<LinearLayout … >
  <TextView … />
  <LinearLayout …
    android:orientation="horizontal" >
    <TextView
      android:background="@color/red50"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_weight="1" />
    <TextView
      android:background="@color/green50"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_weight="1" />
    <TextView … />
    <TextView … />
  </LinearLayout>
</LinearLayout>
```

67

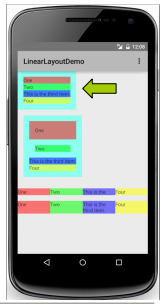
Layout Margin vs. Padding

- *Padding* adds space to a *View* or *View Group* object *inside* its rectangular region
  - Padding is not a layout parameter.
  - The padding space belong to the *View* or *View Group* object itself
- *Layout margin* adds space to a *View* or *View Group* object *outside* its rectangular region
  - Layout margin is a layout parameter
  - The margin space belong to the container

DePaul University 68

Linear Layout Demo – activity_main_demo8.xml

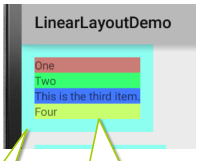
```
<LinearLayout
    android:orientation="vertical"
    android:background="@color/bg01"
    android:padding="16dp
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/red50"
    android:text="@string/one" />
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/green50"
    android:text="@string/two" />
…
```



69

Linear Layout Demo – activity_main_demo8.xml

```
…
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="@color/blue50"
  android:text="@string/three" />
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="@color/yellow50"
  android:text="@string/four" />
</LinearLayout>
```



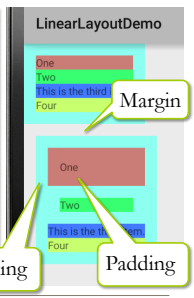
Padding inside the container.

All children are of the same width, the widest.

DePaul University 72

Linear Layout Demo – activity_main_demo8.xml

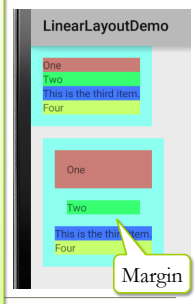
```
<LinearLayout
    android:orientation="vertical"
    android:background="@color/bg01"
    android:layout_margin="16dp"
    android:padding="16dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp"
    android:background="@color/red50"
    android:text="@string/one"/>
…
```



DePaul University 75

Linear Layout Demo – activity_main_demo8.xml

```
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_margin="16dp"
  android:background="@color/green50"
  android:text="@string/two"/>
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="@color/blue50"
  android:text="@string/three"/>
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="@color/yellow50"
  android:text="@string/four"/>
</LinearLayout>
```



77

A Simple Message App – Linear Layout

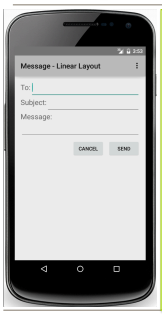
A Simple Message Form

```
<LinearLayout … >
    <LinearLayout … >
        <TextView …/>
        <EditText …/>
    </LinearLayout>
    <LinearLayout …>
        <TextView …/>
        <EditText …/>
    </LinearLayout>
    <TextView …/>
    <EditText …/>
    <LinearLayout …>
        <Button …/>
        <Button …/>
    </LinearLayout>
</LinearLayout>
```

Layout weight: 1

Layout weight: 1

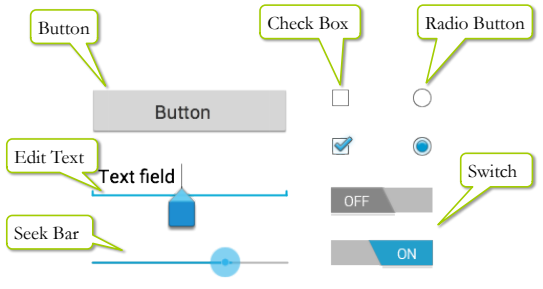
Wrap content
Layout gravity: right



DEPAUL UNIVERSITY 82

Input Controls

Android Common Input Controls



DEPAUL UNIVERSITY 84

Common Input Controls

- *Button*
  - A push-button that can be pressed by the user to perform an action.
- Text input fields
  - *Edit Text*
    - An editable text field.
  - *Auto-Complete Text View*
    - Provides auto-complete suggestions

Next topic

DEPAUL UNIVERSITY 86

Common Input Controls

- *Checkbox*
  - An on/off switch that can be toggled by the user.
  - A group of selectable options that are not mutually exclusive.
- *Radio button*
  - Similar to checkboxes, except that only one option can be selected in the group.
- *Toggle button* and *Switch*
  - An on/off button.

Next lecture

DePaul University 88

Common Input Controls

- *Seek Bar*
  - Interactive slider for selecting a value from a continuous or discrete range of values by moving the slider thumb.
- *Rating Bar*
  - Shows a rating in stars.
  - Can set the rating by touch or drag
- *Progress Bar*
  - Shows percentage of completion

Next lecture

DePaul University 90

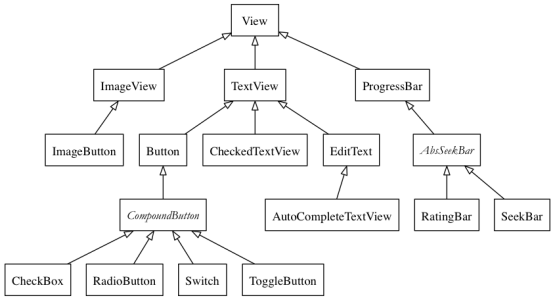
Common Input Controls

- *Spinner*
  - A drop-down list that allows users to select one value from a list.
- Pickers
  - A dialog for users to select a single value
  - *Date Picker*
    - For a date (month, day, year)
  - *Time Picker*
    - For a time (hour, minute, AM/PM)
    - Formatted automatically for the user's locale.

Later

DePaul University 92

The Input Control Classes – Inheritance Relation



DePaul University 93

Text Input Fields

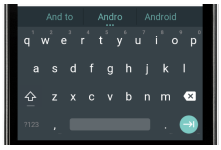
Text Input Fields

- Allow user to input text
  - Single line text (default), or multi-line text
- *Edit Text*
  - An editable text field.
  - Class: `android.widget.EditText`
- *Auto-Complete Text View*
  - Provides auto-complete suggestions
  - Class: `android.widget.AutoCompleteTextView`
    `android.widget.MultiAutoCompleteTextView`

DePaul University 95

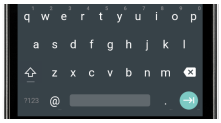
Soft Keyboard Type

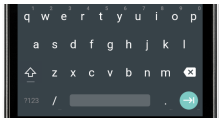
- Supports several different types of soft-keyboard
- Specify the keyboard type using the `android:inputType` attribute
  - `Text`
    Normal text keyboard.



Soft Keyboard Type

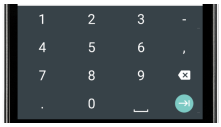
- More values of `android:inputType` attribute
- `textEmailAddress`
  Normal text keyboard with the @ character.
- `textUri`
  Normal text keyboard with the / character.

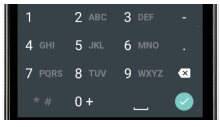




Soft Keyboard Type

- More values of `android:inputType` attribute
- `number`
  Basic number keypad.
- `phone`
  Phone -style keypad.





More Soft Keyboard Types

- Additional input types that control keyboard behavior
  - `textCapSentences` – capitalize the first letter for each new sentence.
  - `textCapWords` – capitalize every word
  - `textAutoCorrect` – corrects commonly misspelled words.
  - `textPassword` – characters displayed as dots.
  - `textMultiLine` – allow long strings of text with line breaks. Default is single line for *Edit Text*.
- Allow bitwise combination: *`op1 | op2`*

Specify Keyboard Actions

- You can specify an action to be performed when the input to a text field is complete.
- Specify the keyboard action using the `android:imeOptions` attribute

Will have an example

- Default actions:
  - Every text field except the last one: `actionNext`, i.e., take the user to the next text field
  - The last text field: `actionDone`, dismiss the keyboard

Common Keyboard Actions

- `actionGo` – *Go*, e.g., entering a URL.
- `actionSearch` – *Search*
- `actionSend` – *Send*, e.g., composing a message.
- `actionNext` – *Next*, taking user to the next text field
- `actionDone` – *Done*, closing the soft keyboard

The Message App – The Layout

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content">
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/to"
    android:textSize="20sp"/>
  <EditText
    android:id="@+id/to"
    android:inputType="textPersonName|textCapWords”
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1” />
</LinearLayout>
```

103

The Message App – The Layout (cont'd)

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content”>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/subject"
    android:textSize="20sp"/>
  <EditText
    android:id="@+id/subject"
    android:inputType="textEmailSubject|textCapWords|textAutoCorrect”
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1”/>
</LinearLayout>
```

DePaul University 104

The Message App – The Layout (cont'd)

```
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/message"
  android:textSize="20sp"/>

<EditText
  android:id="@+id/message"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:inputType="textAutoCorrect” />
```

DePaul University 105

The Message App

DePaul University 111

The Message App – The Action Button

- You can set the action button that is appropriate for the app
- Specify the keyboard action using the `android:imeOptions` attribute

```
<EditText
  android:id="@+id/message"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:inputType="textAutoCorrect”
  android:imeOptions="actionSend" />
```

DePaul University 115

The Message App – Respond to Keyboard Action

- Use an action listener, similar to respond to button action
- Interface `TextView.OnEditorActionListener`
- Action method

```
public boolean onEditorAction(TextView v, int actionId,
                              KeyEvent event)
```

- Return value:
  - true if the event has been handled, false otherwise
- Default actions no longer in effect

The Message App – Respond to Keyboard Action

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  final EditText message = (EditText) findViewById(R.id.message);
  message.setOnEditorActionListener(
    new TextView.OnEditorActionListener() {
      public boolean onEditorAction(TextView v, int actionId,
                                    KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
          sendMessage();
          handled = true;
        }
        return handled;
      }
  });
  …
}
```

Handle Send action only

The Message App – Respond to Keyboard Action

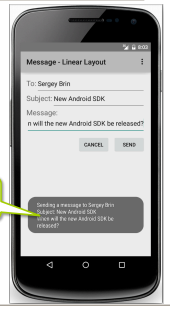
- Use a *Toast* to implement a mock response

```
protected void sendMessage() {
  EditText to = (EditText) findViewById(R.id.to);
  EditText subject = (EditText) findViewById(R.id.subject);
  EditText message = (EditText) findViewById(R.id.message);

  String messageText = "Sending a message to " + to.getText() +
    "\n" + "Subject: " + subject.getText() + "\n" +
    message.getText();
  int duration = Toast.LENGTH_SHORT;
  Toast toast = Toast.makeText(this, messageText, duration);
  toast.show();
}
```


Toast

- A *toast* provides simple feedback about an operation in a small popup.
- A toast is only visible for a limited duration of time.



A toast

Dismiss Keyboard in Action Listener

- When a user defined *On Editor Action Listener* is used, the default editor actions are no longer in effect.
  - e.g., dismissing the keyboard
- You must explicitly dismiss the keyboard in the action listener:

```
InputMethodManager imm = (InputMethodManager)
    getSystemService(Context.INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(message.getWindowToken(), 0);
```


The Message App – Respond to Keyboard Action

```
message.setOnEditorActionListener(
  new TextView.OnEditorActionListener() {
    public boolean onEditorAction(TextView v, int actionId,
                                  KeyEvent event) {
      boolean handled = false;
      if (actionId == EditorInfo.IME_ACTION_SEND) {
        InputMethodManager imm = (InputMethodManager)
            getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(message.getWindowToken(), 0);

        sendMessage();
        handled = true;
      }
      return handled;
    }
});
```

Dismiss the keyboard before handling the action

The Message App – Respond to Button Press

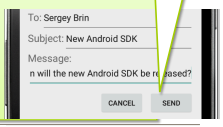
- Add an action listener to the "Send" Button

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

  …
  Button send = (Button) findViewById(R.id.send);
  send.setOnClickListener(new View.OnClickListener() {
    public void onClick (View v) {
      sendMessage();
    }
  });

}
```

The "Send" button



The Message App – Respond to Button Press

- An alternative to handle button action
  - Use the `android:onClick` attribute to specify a method name in the activity class
  - The method must have the signature
    `public void actionName(View view)`
- The "Send" button

```
<Button …
    android:onClick="sendMessage"/>
```


The Message App – Respond to Button Press

- In the activity class

```
protected void sendMessage() {
  …
}

public void sendMessage(View v) {
   sendMessage();
}
```


The Message App – Using Hints

- Hints for text fields
  - Display a prompt when the text field is empty

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content">
  <TextView … />
  <EditText …
     android:hint="@string/hint_to" />
</LinearLayout>
```

Multi-Line Text Field

- *Edit Text* is single line by default
- A multi-line text field

```
<EditText …
  android:inputType="textAutoCorrect|textMultiLine"/>
```

  - Starts with a single line
  - Expands to multi-line as needed
- For a multi-line text field, the keyboard action button is the "*New Line*" key
  - Not redefinable

Auto-Complete Text Fields

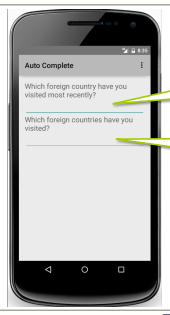
Auto-Complete Text View

- Class *Auto-Complete Text View*
  - A subclass of *Edit Text*
- An editable text view that shows completion suggestions automatically while the user is typing.
- The text suggestions are provided through an *adapter*.
  - A bridge between data items and view objects
  - *Array Adapter*
    - An adapter accepts data in an array or a list

Multi Auto-Complete Text View

- Class *Multi Auto Complete Text View*
  - A subclass of *Auto Complete Text View*
- Shows completion suggestions for the substring of the text
- Requires a tokenizer to separate the substrings
  - A *Comma Tokenizer* is provided
    - A simple tokenizer for substrings separated by comma (,) followed by space(s)

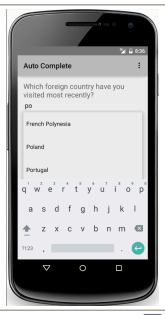
The Auto-Complete App

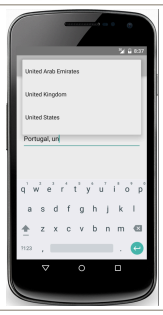


Auto Complete Text View for the name of a country

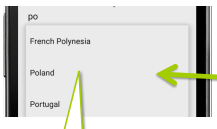
Multi Auto Complete Text View for a list of country names

The Auto-Complete App





The Role of the Adapter



More on adapters later

Adapter

String data

View objects

…
"French Polynesia"
…
"Poland"
…
"Portugal"
…

The Auto-Complete App – The Layout

```
<LinearLayout … >
  <TextView … />
  <AutoCompleteTextView
    android:id="@+id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
  <TextView … />
  <MultiAutoCompleteTextView
    android:id="@+id/text2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

The Auto-Complete App – Provide the Suggestions

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
      android.R.layout.simple_dropdown_item_1line, COUNTRIES);
  AutoCompleteTextView tv1 =
    (AutoCompleteTextView) findViewById(R.id.text1);
  tv1.setAdapter(adapter);
  …
}

static final String[] COUNTRIES = new String[] {
  "Afghanistan", "Albania", "Algeria", …
  "Zambia", "Zimbabwe"
};
```

What is android.R?

android.R

- A framework class
  - Provides access to framework defined resources
  - Available system-wide
- Not to be confused with the project-specific **R.java**

The Auto-Complete App – Provide the Suggestions

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, COUNTRIES);
  AutoCompleteTextView tv1 =
      (AutoCompleteTextView) findViewById(R.id.text1);
  tv1.setAdapter(adapter);
  …
}

static final String[] COUNTRIES = new String[] {
  "Afghanistan", "Albania", "Algeria", …
  "Zambia", "Zimbabwe"
};
```


Auto-Complete App – The Activity

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, COUNTRIES);
  …
  MultiAutoCompleteTextView tv2 =
      (MultiAutoCompleteTextView) findViewById(R.id.text2);
  tv2.setAdapter(adapter);
  tv2.setTokenizer(
      new MultiAutoCompleteTextView.CommaTokenizer());
}
```

Use the system defined tokenizer

The Sample Code

- The sample apps in this lecture are available in D2L
  - `HelloDrawable.zip`
  - `LinearLayoutDemo.zip`
  - `Message-LinearLayout.zip`
  - `AutoComplete.zip`
- Each zip archive contains the entire project folder
- Unzip the files and import to *Android Studio*

Next …

- More UI widgets
  - Radio buttons, checkboxes, toggle buttons, switches
  - Seek bars, rating bars
  - Image views and image buttons
- Intents and intent filters
- Multi-screen apps
- Activity lifecycle