20CSI605 - Mobile Application Development

Delightful user experience

1.Drawables, Styles and Themes

Drawables

- A *drawable is a graphic that can be drawn to the screen. You retrieve drawables using APIs such as getDrawable(int) ,*
- and you apply a drawable to an XML resource using attributes such as android:drawable and android:icon .

Using drawables

To display a Drawable, use the ImageView class to create a View. In the <ImageView> element in your XML file, define how the Drawable is displayed and where the Drawable file is located.

For example, this ImageView displays an image called "birthdaycake.png":

1.Drawables, Styles and Themes

Using drawables

```
<ImageView
      android:id="@+id/tiles"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/birthdaycake" />
```

- The android:id attribute sets a shortcut name that you use to call the image later.
- The android:layout_width and android:layout_height attributes specify the size of the View.
- The android:src attribute gives the location where this image is stored
- <ImageView> also has attributes that you can use to crop your image

1.Drawbacks, Styles and Themes

Image files

- An image file is a generic bitmap file. Android supports image files in several formats: WebP (preferred), PNG (preferred), and JPG (acceptable). GIF and BMP formats are supported, but discouraged.
- The WebP format is fully supported from Android 4.2. WebP compresses better than other formats for lossless and lossy compression, potentially resulting in images more than 25% smaller than JPEG formats.
- You can convert existing PNG and JPEG images into WebP format before upload. For more about WebP, see the WebP documentation.

1.Drawbacks, Styles and Themes

Nine-patch files

- A *9-patch* is a PNG image in which you define stretchable regions. Use a 9-patch as the background image for a View to make sure the View looks correct for different screen sizes and orientations.
- The Android standard Button is an example of a View that uses a 9-patch as its background image. The 9-patch stretches to accommodate the text or image inside the Button.
- Save 9-patch files with a .9.png extension and store them in the res/drawable folder.
- Use them with the android:src attribute for an ImageView and its descendants, or to create a NinePatchDrawable class in Java code

1.Drawbacks, Styles and Themes

To create a 9-patch, use the Draw 9-Patch tool in Android Studio

To use the tool:

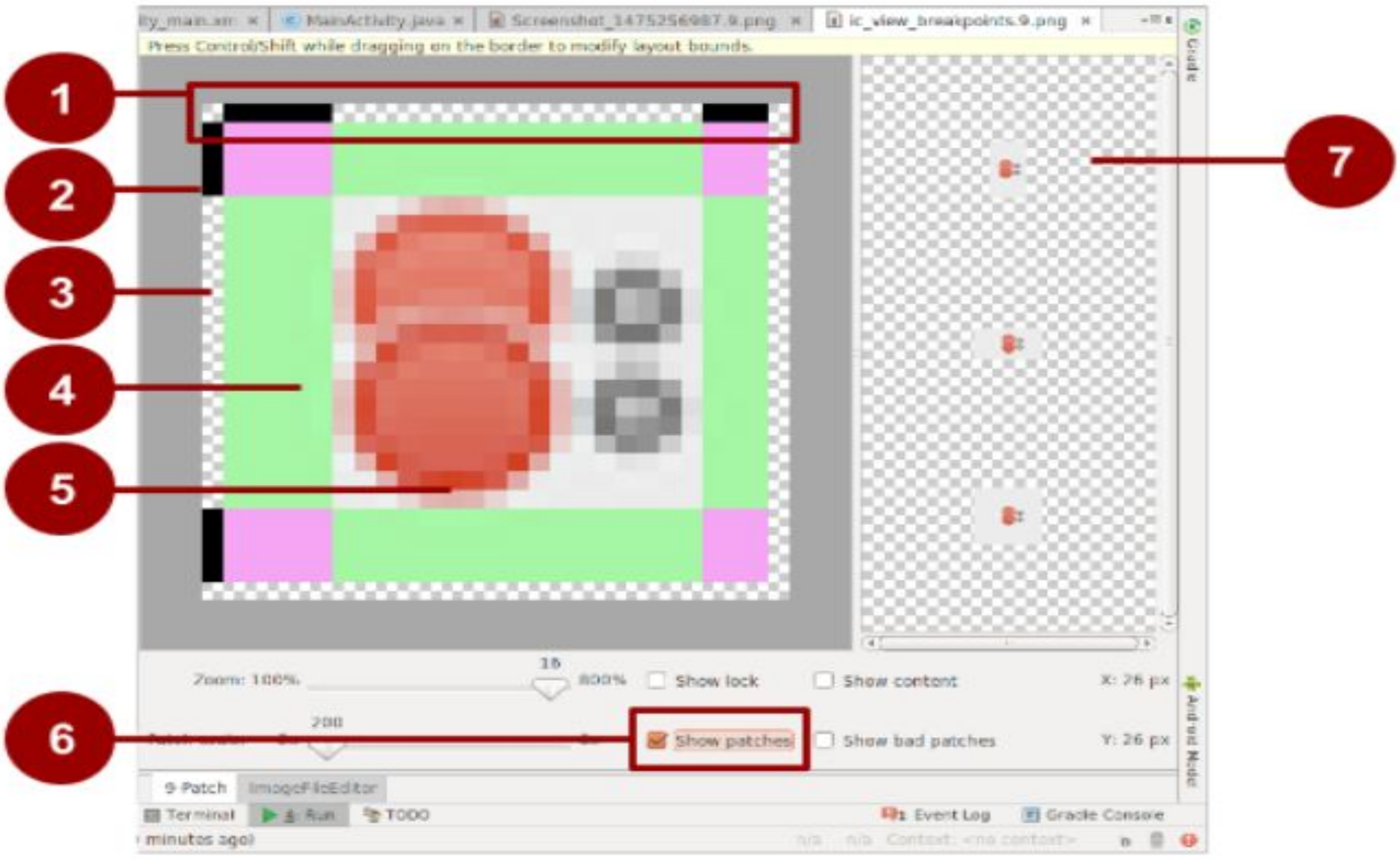
1.Put a PNG file into the res/drawable folder. (To do this, copy the image file into the **app/src/main/res/drawable** folder of your project.)

2.In Android Studio, **right-click** (or **Control-click**) the file and choose **Create 9-Patch file**. Android Studio saves the file with a .9.png extension.

3. In Android Studio, double-click the **.9.png** file to open the editor.

4. Specify which regions of the image are okay to stretch.

1.Drawbacks, Styles and Themes



Press Control/Shift while dragging on the border to modify layout bounds.

Zoom: 100% 16 800% Show lock Show content X: 26 px

Patch scale: 2x 200 6x [✓ Show patches] Show bad patches Y: 26 px

minutes ago)

1.Drawbacks, Styles and Themes

1. Border to indicate which regions are okay to stretch for width (horizontally).
2. Border to indicate regions that are okay to stretch for height (vertically).
3. Turn off pixels by pressing Shift-click (Control-click in macOS).
4. Stretchable area.
5. Not stretchable.
6. Check Show patches to preview the stretchable patches in the drawing area.
7. Previews of stretched image.

1.Drawbacks, Styles and Themes

Level list drawables

- ✓ A level list drawable defines alternate drawables, each assigned a maximum numerical value.
- ✓ To select which drawable to use, call the setLevel() method, passing in an integer that is matched against the maximum level integer defined in XML.
- ✓ The resource with the lowest maximum level greater than or equal to the integer passed into setLevel() is selected.

1.Drawbacks, Styles and Themes

Transition drawables

- A TransitionDrawable is a Drawable that crossfades between two other drawables.
- To define a TransitionDrawable in XML, use the <transition> element. Each Drawable is represented by an <item> element inside the <transition> element.
- No more than two <item> elements are supported.

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

1.Drawbacks, Styles and Themes

Images

- ✓ Images, from launcher icons to banner images, are used in many ways in Android.
- ✓ Each use case has different requirements for image resolution, scalability and simplicity.
- ✓ In this section you learn about the different ways to generate images and include them in your app.

Creating icons

Every app requires at least a launcher icon, and apps often include icons for action bar actions, notifications, and other use cases.

1.Drawbacks, Styles and Themes

Creating icons

There are two approaches to creating icons:

- Create a set of image files of the same icon in different resolutions and sizes so that the icon looks the same across devices with different screen densities. You can use Image Asset Studio to do this.
- Use vector drawables, which scale automatically without the image becoming pixelated or blurry. You can use Vector Asset Studio to do this.

1.Drawbacks, Styles and Themes

Styles

- In Android, a style is a collection of attributes that define the look and format of a View.
- You can apply the same style to any number of View elements in your app; for example, several TextView elements might have the same text size and layout.
- Using styles allows you to keep these common attributes in one location and apply them to each TextView using a single line of code in XML.
- You can define styles yourself or use one of the platform styles that Android provides

1.Drawbacks, Styles and Themes

Styles

To create a style, add a <style> element inside a <resources> element in any XML file located in the values folder inside the res folder in the Project > Android pane. When you create a project in Android Studio, a styles.xml file is created for you.

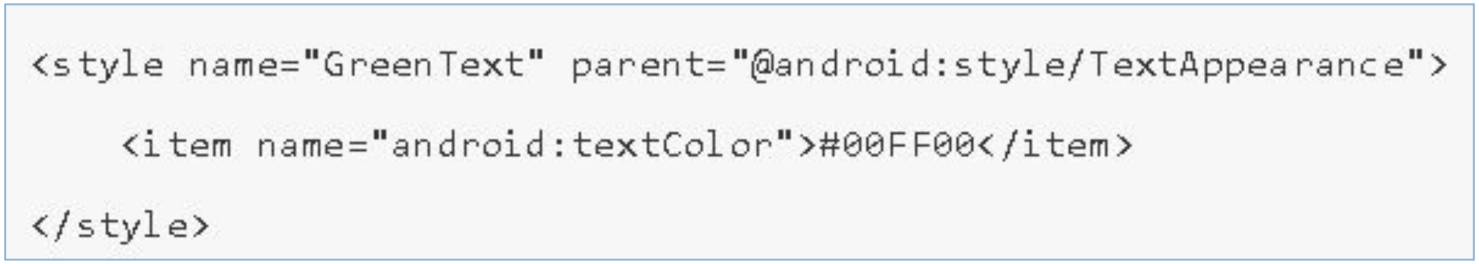
A <style> element includes the following:

- ✔ A name attribute. Use the style's name when you apply the style to a View.
- ✔ An optional parent attribute. You learn about using parent attributes in the Inheritance section below.
- ✔ Any number of <item> elements as child elements of <style>. Each <item> element includes one style attribute.

1.Drawbacks, Styles and Themes

Inheritance

- ✔ A new style can inherit the properties of an existing style.
- ✔ When you create a style that inherits properties, you define only the properties that you want to change or add.
- ✔ You can inherit properties from platform styles and from styles that you create yourself.
- ✔ **To inherit a platform style**, use the parent attribute to specify the resource ID of the style you want to inherit.



```
<style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>
```

1.Drawbacks, Styles and Themes

<u>Themes</u>

- ✔ You create a theme the same way you create a style, which is by adding a <style> element inside a <resources> element in any XML file located in the values folder inside the res folder in the **Project > Android** pane.

What's the difference between a style and a theme?

- ✔ A style applies to a View. In XML, you apply a style using the style attribute.
- ✔ A theme applies to an Activity or an entire app, rather than to an individual View. In XML, you apply a theme using the android:theme attribute.

1.Drawbacks, Styles and Themes

Applying Themes

- To apply a theme to your app, declare it inside an <application> element inside the AndroidManifest.xml file. This example applies the AppTheme theme to the entire application:

```
android:theme="@style/AppTheme"
```

- To apply a theme to an Activity, declare it inside an <activity> element in the AndroidManifest.xml file. In this example, the android:theme attribute applies the Theme_Dialog platform theme to the Activity:

```
<activity android:theme="@android:style/Theme.Dialog">
```

1.Drawbacks, Styles and Themes

Default theme

- ✔ When you create a new project in Android Studio, a default theme is defined for you within the styles.xml file.

For example, this code might be in your styles.xml file:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

2.Material Design

- Material Design is a visual design philosophy that Google created in 2014.
- The aim of Material Design is a unified user experience across platforms and device sizes.
- Material Design includes a set of guidelines for style, layout, motion, and other aspects of app design.
- Material Design is for desktop web applications as well as for mobile apps.

2.Material Design

Principles of Material Design

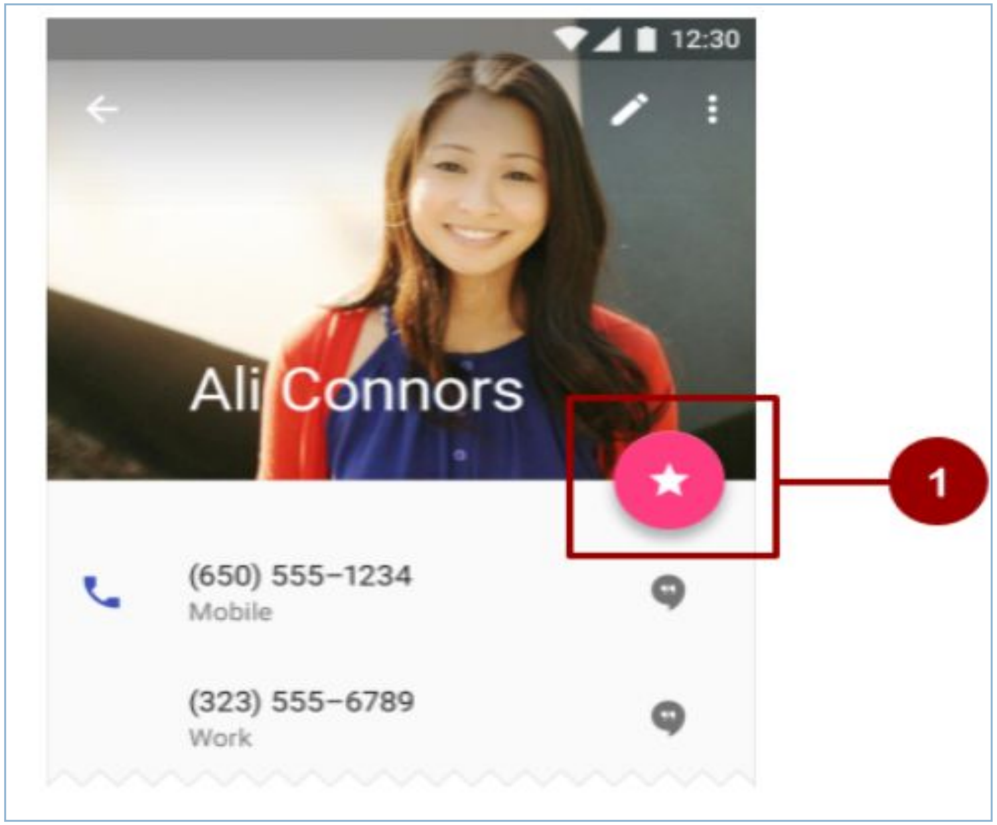
- ✔ In Material Design, elements in your Android app behave like real world materials: they cast shadows, occupy space, and interact with each other.

Bold, graphic, intentional

- Material Design involves deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space that create a bold and graphic interface.
- Emphasize user actions in your app so that the user knows right away what to do, and how to do it.

2.Material Design

For example, highlight things that users can interact with, such as buttons, EditText fields, and switches.



#1 is a `FloatingActionButton`

2.Material Design

The following are the material design used for:

- ✓Colors
- ✓Typography
- ✓Layout
- ✓Components and patterns
- ✓Motion

3. Resources for adaptive layouts

An adaptive layout is a layout that works well for different screen sizes and orientations, different devices, different locales and languages, and different versions of Android.

Externalizing resources

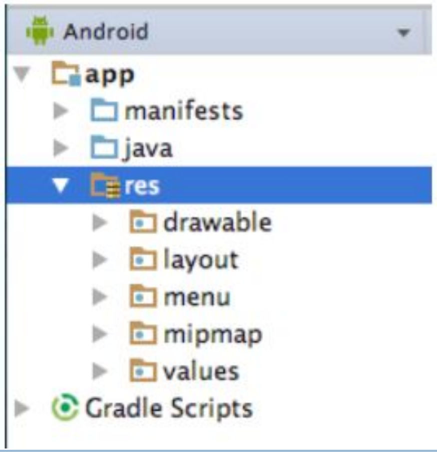
Always externalize resources such as drawables, icons, layouts, and strings.

- ✓ You can maintain externalized resources separately from your other code.
- ✓ You can provide alternative resources that support specific device configurations, for example devices with different languages or screen sizes.

3. Resources for adaptive layouts

Grouping resources

- Store all your resources in the res folder. Organize resources by type into folders within res. You must use standardized names for these folders.
- For example, the screenshot below shows the file hierarchy for a small project, as seen in the Project > Android pane. The folders that contain this project's default resources use standardized names: drawable, layout, menu, mipmap (for icons), and values.



THANK YOU