2.2 User Interfaces

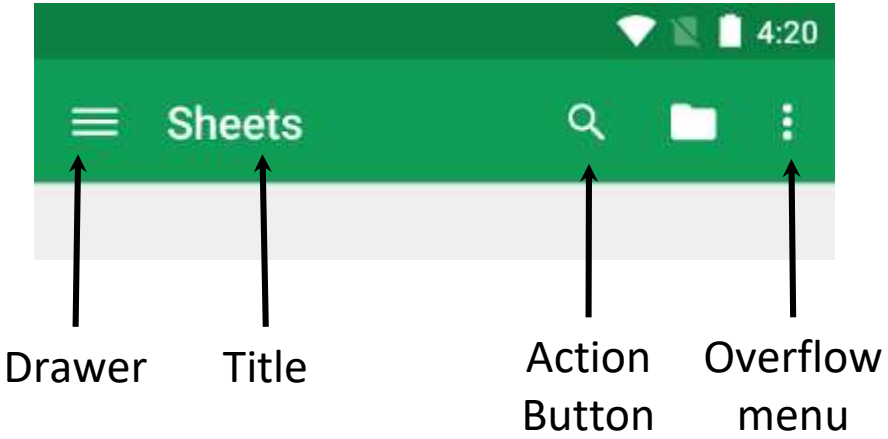
Menus, Notifications and Design for everyone

Objectives

- Learn to create Menus (ToolBar)
- Learn to use Toasts and Notifications
- Learn to design UI for everyone

Menus

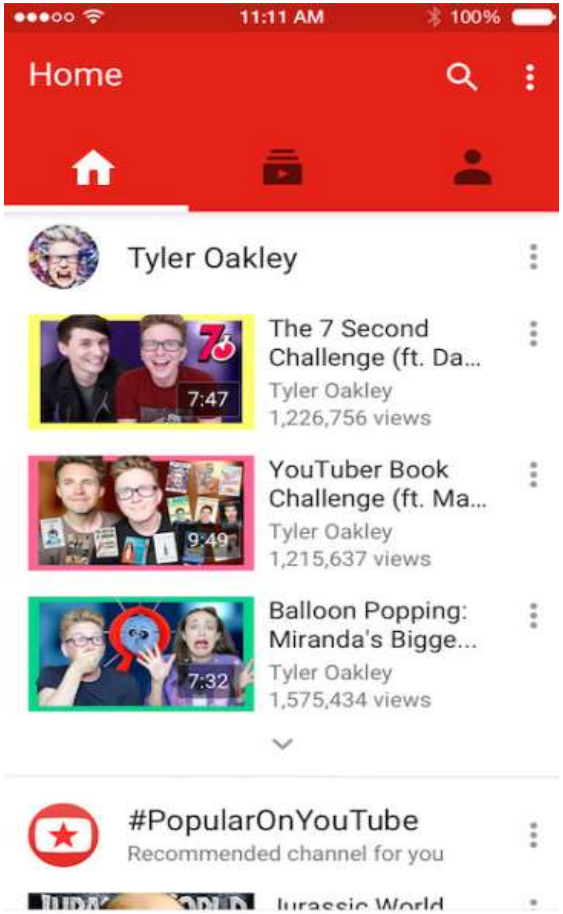
- Presents actions and options to user
- Menus are presented using the Toolbar widget (New in the support library) or Action Bar (Old, does not support material design)

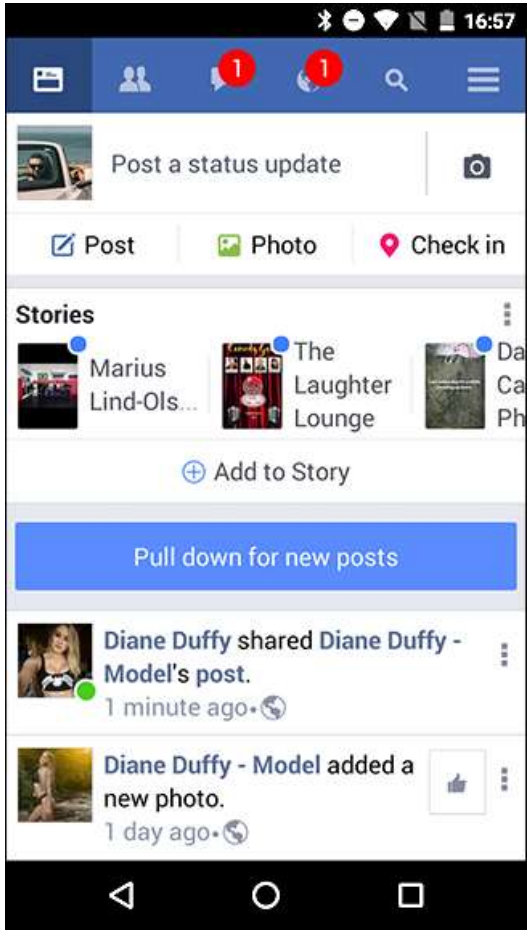


https://developer.android.com/training/appbar

Question?

Why use the Menus?

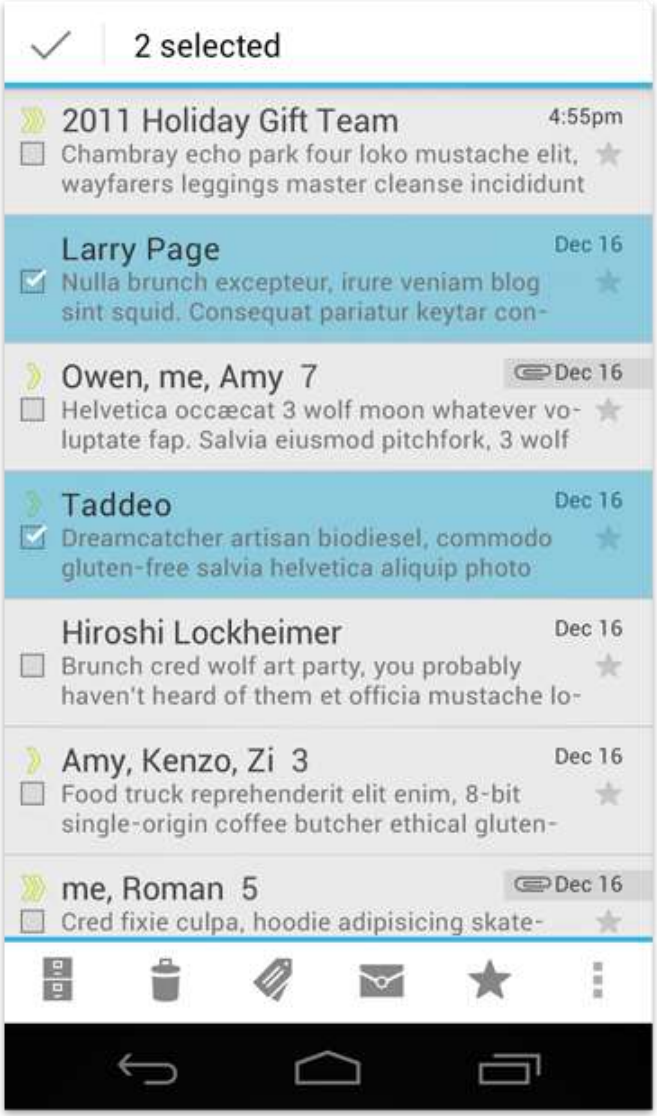




Menus

- For guidance on prioritizing actions, use the FIT scheme:
  a. Frequent
  b. Important
  c. Typical

Context Sensitive Menus



Setting Up the Toolbar

- Basic form = app title + overflow menu
- ToolBar class provides material design
- Activity extends AppCompatActivity class

Kotlin

```
class MyActivity : AppCompatActivity() {
  // ...
}
```

Java

```
public class MyActivity extends AppCompatActivity {
  // ...
}
```

Specify the Actions in XML

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <!-- "Mark Favorite", should appear as action button if possible -->
    <item
        android:id="@+id/action_favorite"
        android:icon="@drawable/ic_favorite_black_48dp"
        android:title="@string/action_favorite"
        app:showAsAction="ifRoom"/>

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never"/>

</menu>
```

Download action bar icons

- To best match the Android iconography guidelines, you should use icons provided in the Material icons site

- https://material.io/icons/

Adding an Up Button

- Up button helps users to find their way back to the app's main screen or parent activity

- A child activity must declare its parent in the <u>manifest file</u>

https://developer.android.com/training/appbar/up-action

Add Up Button for Low-level Activities

```
<activity
        android:name="com.example.myfirstapp.MyChildActivity"
        android:label="@string/title_activity_child"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >

        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
</activity>
```

Enable the Up Button

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my_child);

    // my_child_toolbar is defined in the layout file
    Toolbar myChildToolbar = (Toolbar) findViewById(R.id.my_child_toolbar);
    setSupportActionBar(myChildToolbar);

    // Get a support ActionBar corresponding to this toolbar
    ActionBar ab = getSupportActionBar();

    // Enable the Up button
    ab.setDisplayHomeAsUpEnabled(true);
}
```

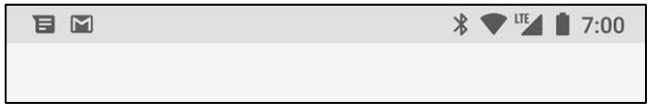
Notification

- A message displays outside your app's UI

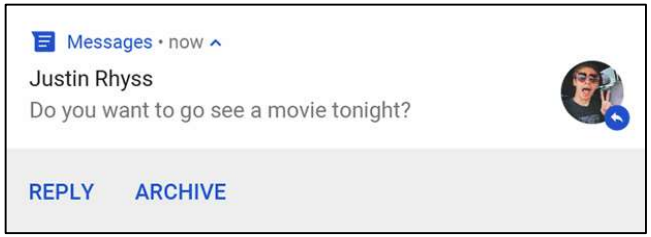
- Purposes:
  - Reminder
  - Communication from other people
  - Other timely information from your app

https://developer.android.com/guide/topics/ui/notifiers/notifications

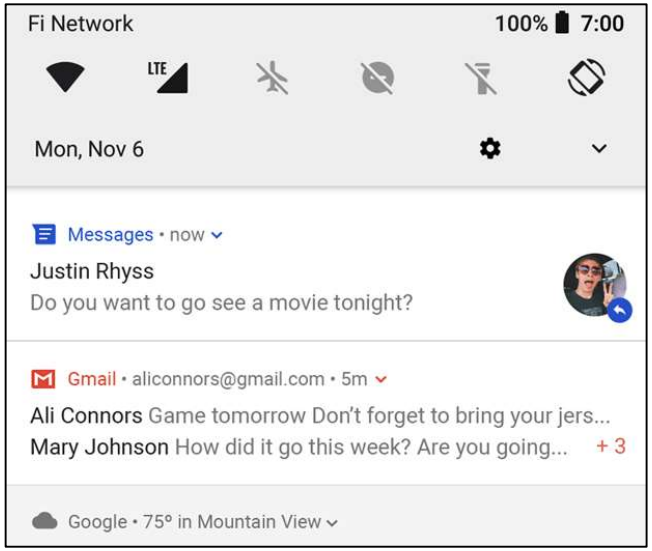
Notification



Notification icons appear on the left side of the status bar



A heads-up notification appears in front of the foreground app



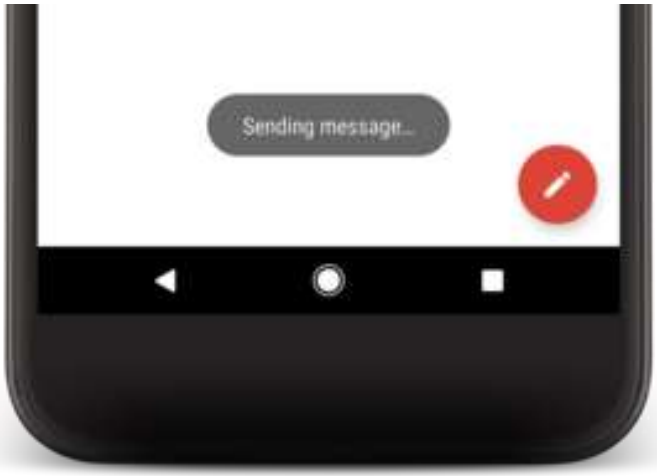
Notifications in the notification drawer

Find out ways to create a notification here : https://developer.android.com/training/notify-user/build-notification

Toast

- It provides simple feedback about an operation in a small popup.

- It only fills the amount of space required for the message.



https://developer.android.com/guide/topics/ui/notifiers/toasts

Toast

Kotlin

```
val text = "Hello toast!“

Toast.makeText(applicationContext, text, Toast.LENGTH_SHORT).show()
```

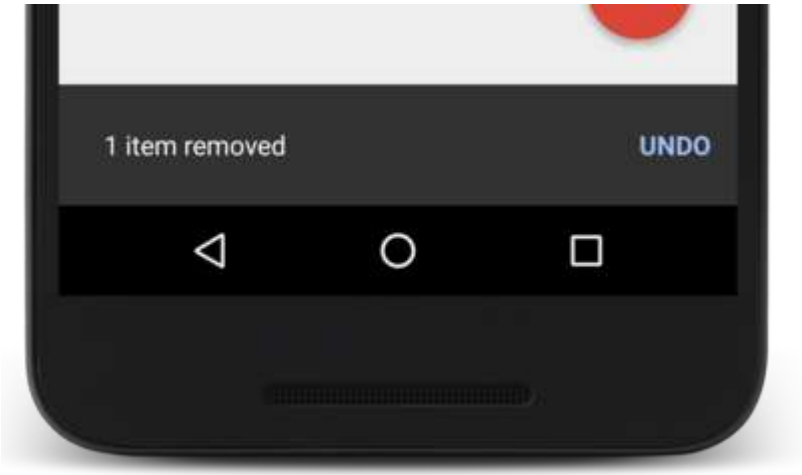
Java

```
CharSequence text = "Hello toast!";

Toast toast = Toast.makeText(getApplicationContext(),
                             text,
                             Toast.LENGTH_SHORT).show();
```

Snackbar

- The Snackbar class supersedes Toast
- Displays a brief message
- Action could be added to snake bar



https://developer.android.com/training/snackbar/action

Snackbar

Kotlin

```kotlin
val mySnackbar = Snackbar.make(findViewById(R.id.myCoordinatorLayout),
                               R.string.email_archived,
                               Snackbar.LENGTH_SHORT)

mySnackbar.setAction(R.string.undo_string, MyUndoListener())

mySnackbar.show()
```

Java

```java
Snackbar mySnackbar = Snackbar.make(
                      findViewById(R.id.myCoordinatorLayout),
                      R.string.email_archived,
                      Snackbar.LENGTH_SHORT);

mySnackbar.setAction(R.string.undo_string,
                     new MyUndoListener());

mySnackbar.show();
```

Question?

1. What is the main difference between a Toast and a Notification? When a Toast is more appropriate?
2. Among Notification and Snackbar, which one should be used to perform the following tasks:
   a. Inform user that the phone internal storage is low
   b. A message has been deleted from In box
   c. An update has been made available

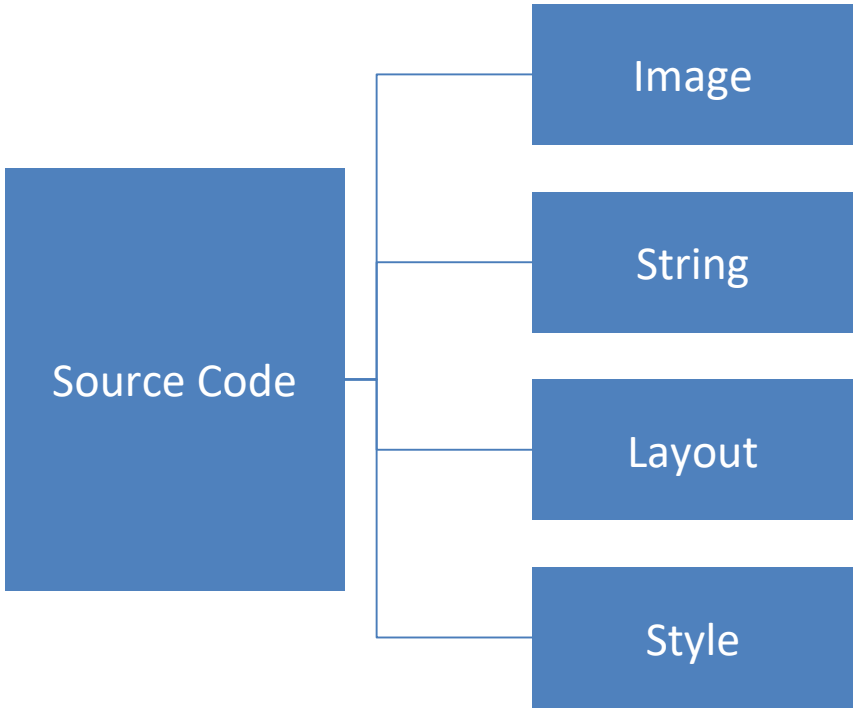
Design for Everyone

- The challenges:
  - Android devices come in many shapes and sizes all around the world
  - Device configurations:
    - Languages
    - Screen sizes, and
    - Versions of the Android platform

https://developer.android.com/guide/practices/compatibility

Techniques

- Externalize resources
- Reason: able maintain resources independently



Supporting Different Languages

- Extract UI strings from your app code and keep them in an external file.

- The res/values/strings.xml holds your string values.

Create Locale Directories and String Files

- Create additional values directories inside res/ that include a hyphen and the ISO country code at the end of the directory name.

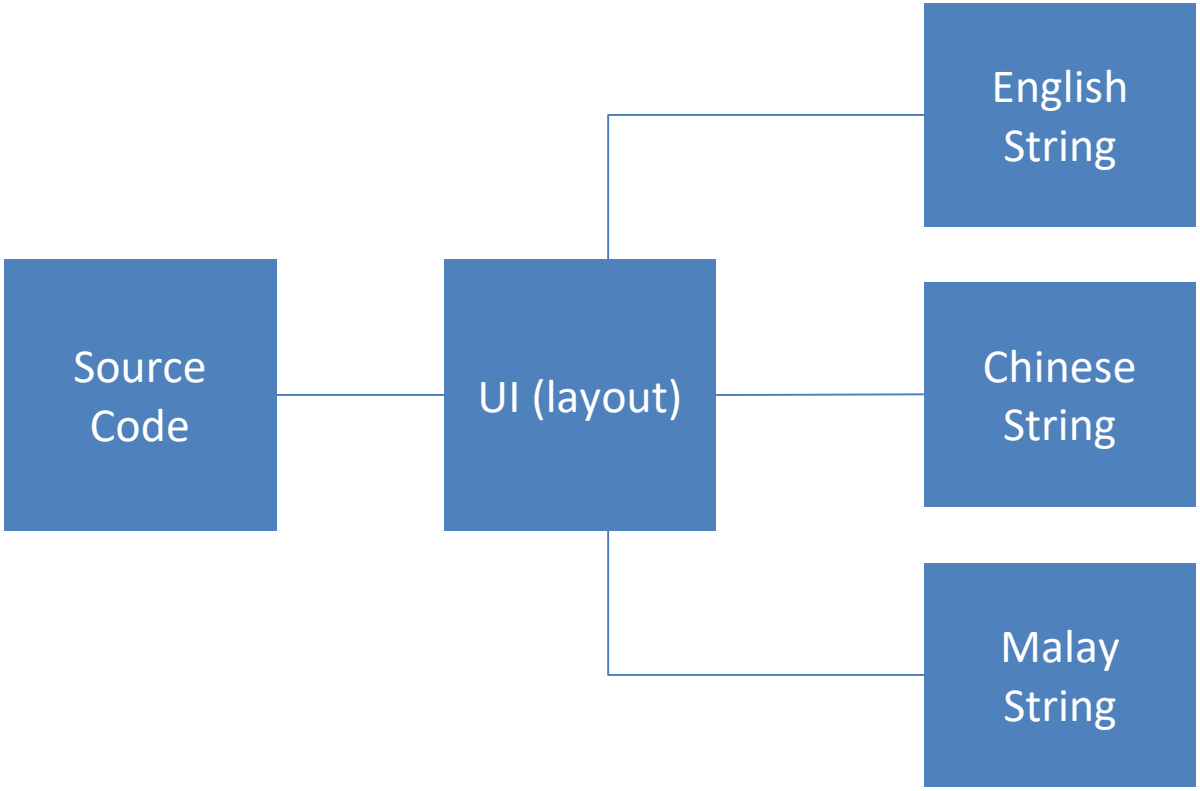
```
MyProject/
  res/
    values/
        strings.xml
    values-es/
        strings.xml
    values-fr/
        strings.xml
```

Alternative Resources (Update)

```
MyProject/
    res/
        values/
            strings.xml
        values-b+es/
            strings.xml
        mipmap/
            country_flag.png
        mipmap-b+es+ES/
            country_flag.png
```

Create Locale Directories and String Files

- At runtime, the Android system uses the appropriate set of string resources based on the locale currently set for the user's device.



Create Locale Directories and String Files

English (default locale), `/values/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

French, `/values-fr/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Language Code

- Locale Language Code:
  - English: en
  - English (UK): en_GB
  - English (US): en_US

  - Chinese language: zh
  - Chinese (China) : zh_CN

  - Malay (Indonesia) : in

  - Bahasa Melayu : ms

Use the String Resources

- In your source code, you can refer to a string resource with the syntax R.string.<string_name>.

Kotlin

```
val hello: String = getString(R.string.hello)
```

Java

```
String string = getString(R.string.hello);
```

Use the String-Array Resources

- You can refer to a string-array resource with the syntax R.string.<string_name>.

```
<resources>
    <string-array name="states_array">
        <item>Johor</item>
        ...
    </string-array>
</resources>
```

Kotlin

```
val array: Array = resources.getStringArray(R.array.planets_array)
```

Java

```
Resources res = getResources();
String[] states = res.getStringArray(R.array.states_array);
```

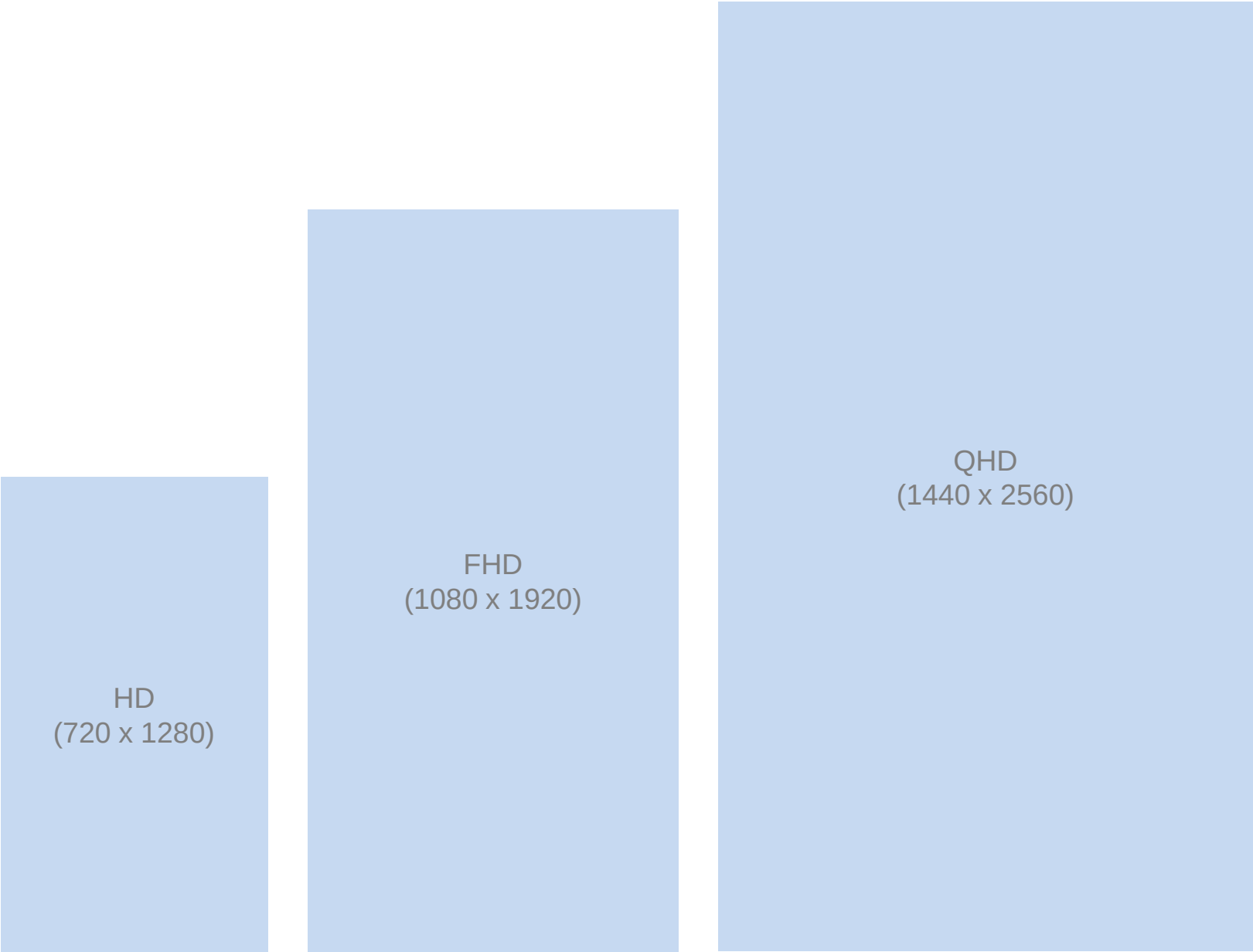
Use the String Resources

- In other XML files, you can refer to a string resource with the syntax @string/<string_name> whenever the XML attribute accepts a string value.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Supporting Different Screens

- Android categorizes device screens using two general properties:
  - **Size**: small, normal, large, xlarge
  - **Density**: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)


HD
(720 x 1280)
FHD
(1080 x 1920)
QHD
(1440 x 2560)

Support Display Cutouts

- Android 9 (API level 28)

- Cutout Mode
  - Default
  - Short Edges
  - Never


```
<style name="ActivityTheme">
  <item name="android:windowLayoutInDisplayCutoutMode">
    shortEdges <!-- default, shortEdges, never -->
  </item>
</style>
```

Short Edge

Short Edge

Never

Supporting Different Screens

- Declare different layouts and bitmaps for different screens
- Place these alternative resources in separate directories
- Screens <u>orientation</u> (landscape or portrait) is considered a variation of screen size

Create Different Layouts

- Create a unique layout XML file for each screen size
- Each layout should be saved into the appropriate resources directory, named with a -<screen_size> suffix.

```
MyProject/
  res/
      layout/
          main.xml
      layout-large/
          main.xml
```

Create Different Layouts

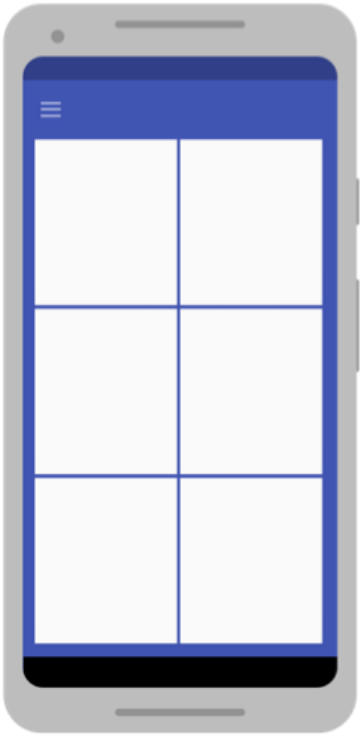
- The file names must be exactly the same, but their contents are different in order to provide an optimized UI for the corresponding screen size.

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

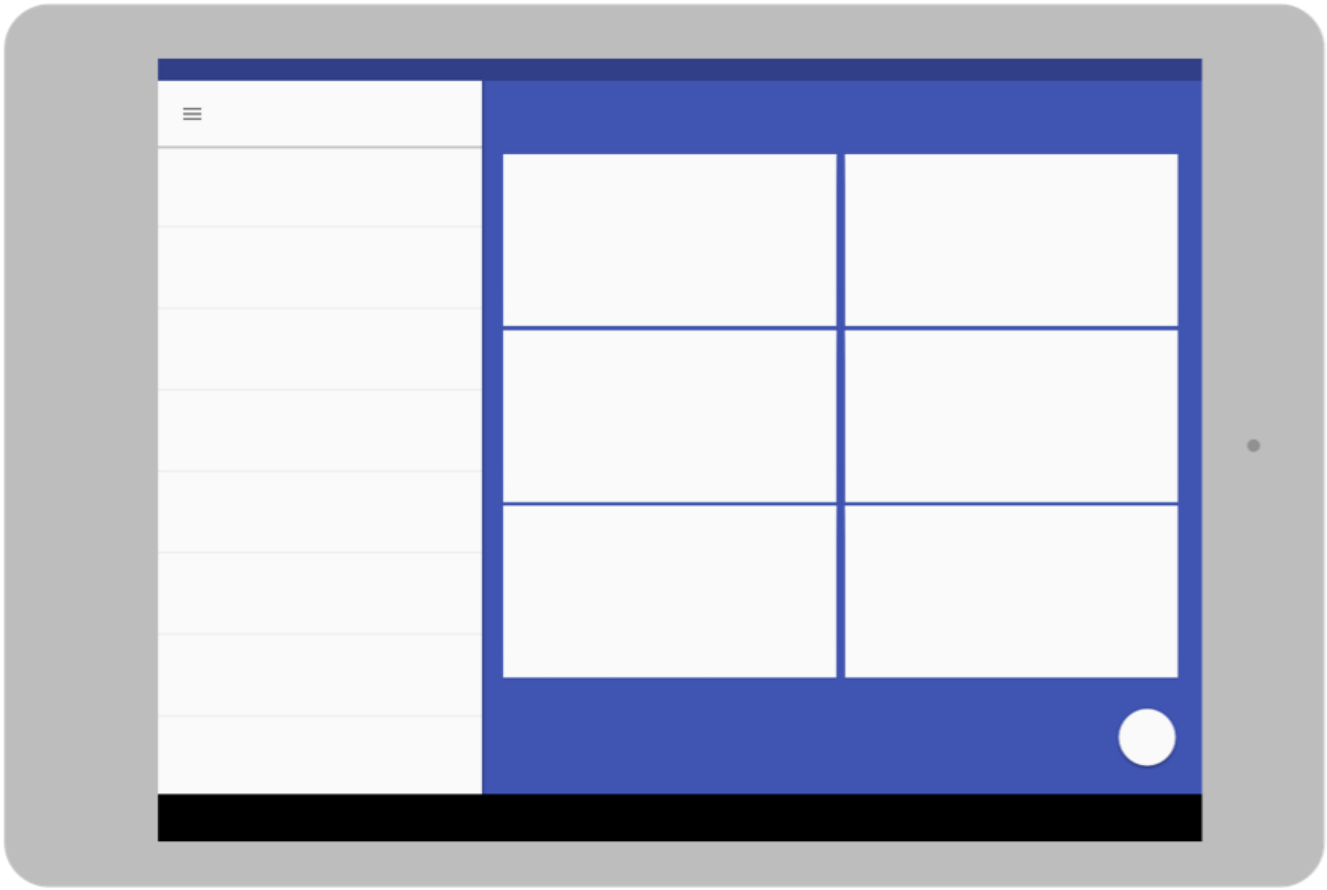
Create Different Layouts

- The system loads the layout file from the appropriate layout directory based on screen size/orientation of the device on which your app is running.

```
MyProject/
   res/
     layout/ # default (portrait)
         main.xml
     layout-land/ # landscape
         main.xml
     layout-large/ # large (portrait)
         main.xml
     layout-large-land/ # large landscape
         main.xml
```



layout/



layout-land/

Create Different Bitmaps

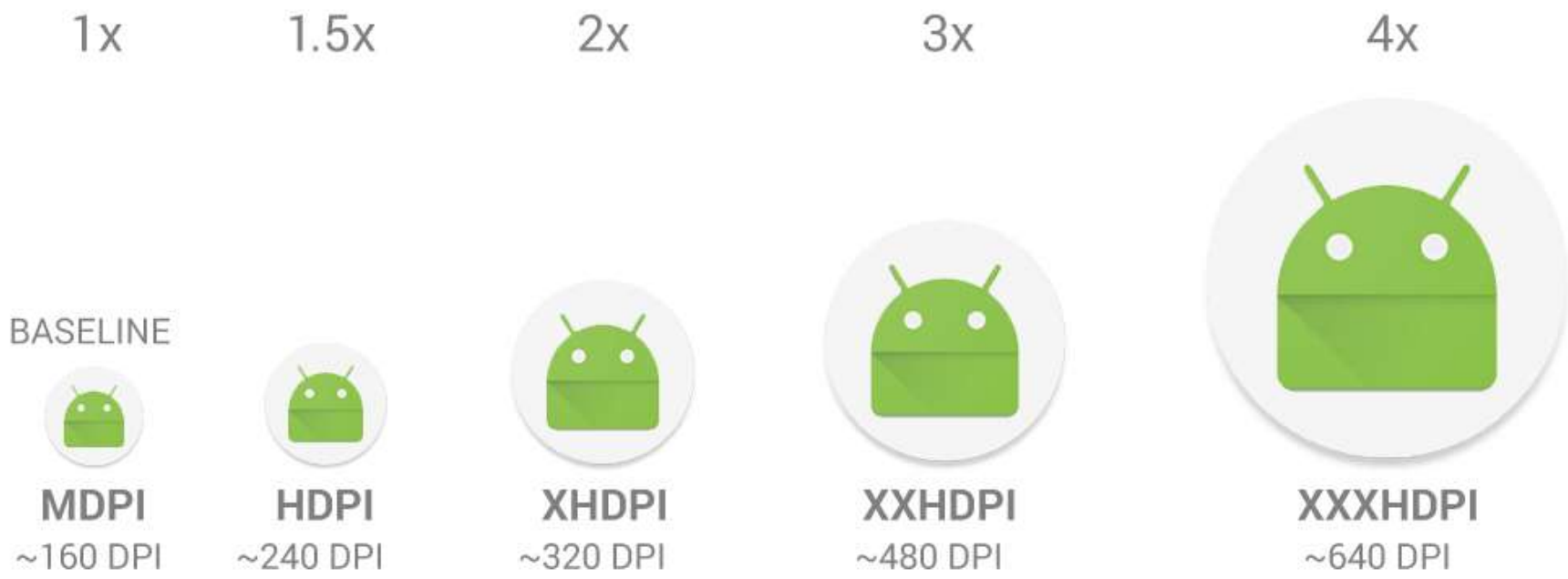
- Provide bitmap resources that are properly scaled to each of the generalized density buckets: low, medium, high and extra-high density.

- It helps to achieve good graphical quality and performance on all screen densities.

Create Different Bitmaps

- Start with raw resource in vector format and generate the images for each density using the following size scale:
  - xhdpi: 2.0
  - hdpi: 1.5
  - mdpi: 1.0 (baseline)

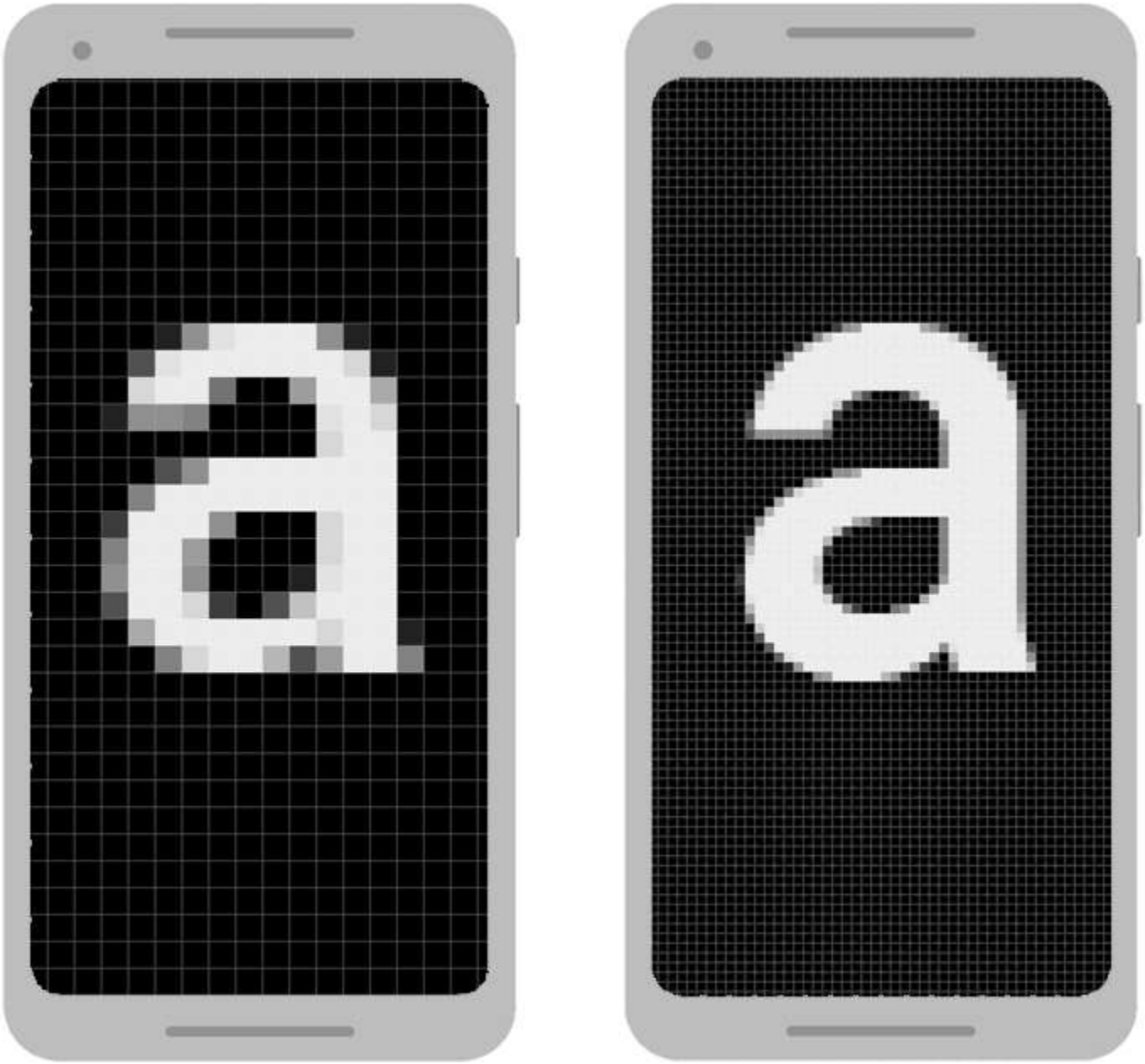
- 200x200 for xhdpi, 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi


1x
1.5x
2x
3x
4x
BASELINE
MDPI
~160 DPI
HDPI
~240 DPI
XHDPI
~320 DPI
XXHDPI
~480 DPI
XXXHDPI
~640 DPI

Create Different Bitmaps

- Place the files in the appropriate drawable resource directory:

```
MyProject/
   res/
     drawable-xhdpi/
        awesomeimage.png
     drawable-hdpi/
        awesomeimage.png
     drawable-mdpi/
        awesomeimage.png
     drawable-ldpi/
        awesomeimage.png
```



Specify Minimum and Target API Levels

- The AndroidManifest.xml file describes details about your app and identifies which versions of Android it supports.

- In the uses-sdk element
  - minSdkVersion identifies the lowest API level with which your app is compatible
  - targetSdkVersion is the highest API level against which you've designed and tested your app

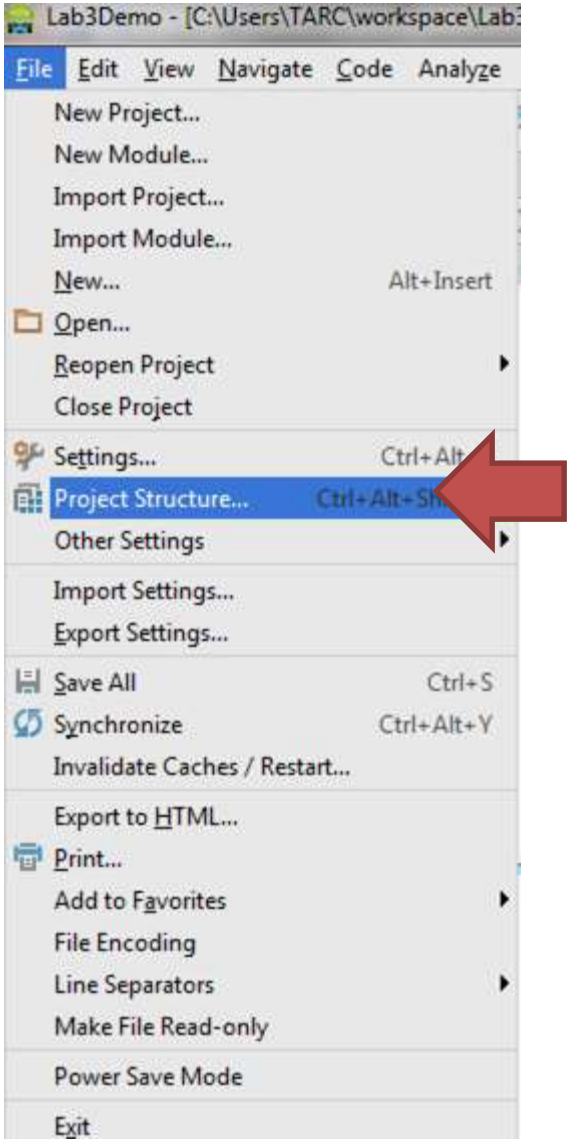
Specify Minimum and Target API Levels

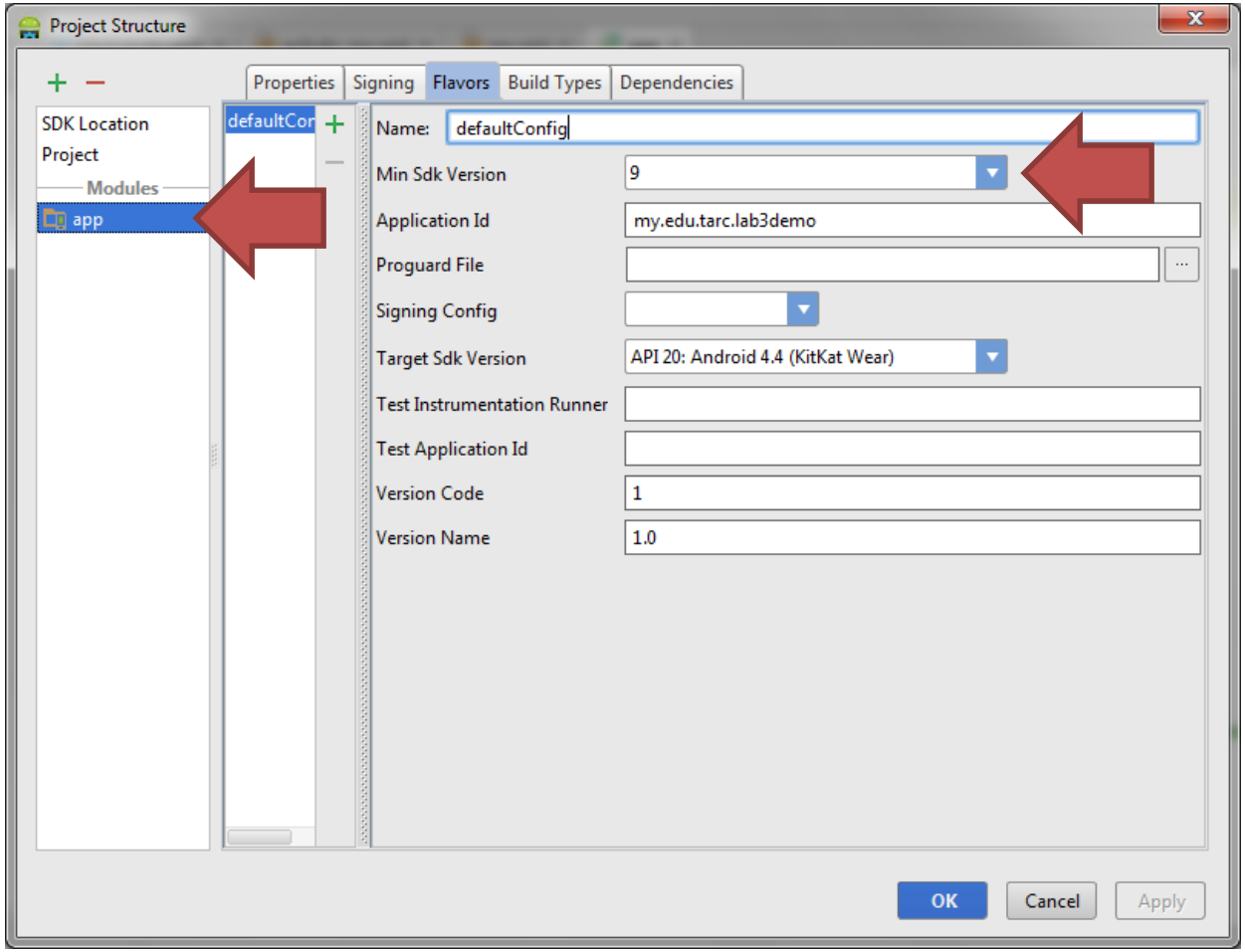
- To allow your app to take advantage of changes and ensure that your app fits the style of each user's device, you should set the `targetSdkVersion` value to match the latest Android version available.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >

    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="28" />
    ...
</manifest>
```

Change SDK


Lab3Demo - [C:\Users\TARC\workspace\Lab3
File
Edit
View
Navigate
Code
Analyze
New Project...
New Module...
Import Project...
Import Module...
New... Alt+Insert
Open...
Reopen Project
Close Project
Settings... Ctrl+Alt
Project Structure... Ctrl+Alt+Sh
Other Settings
Import Settings...
Export Settings...
Save All Ctrl+S
Synchronize Ctrl+Alt+Y
Invalidate Caches / Restart...
Export to HTML...
Print...
Add to Favorites
File Encoding
Line Separators
Make File Read-only
Power Save Mode
Exit


Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
SDK Location
Project
Modules
app
defaultCon
Name: defaultConfig
Min Sdk Version 9
Application Id my.edu.tarc.lab3demo
Proguard File
Signing Config
Target Sdk Version API 20: Android 4.4 (KitKat Wear)
Test Instrumentation Runner
Test Application Id
Version Code 1
Version Name 1.0
OK
Cancel
Apply

Check System Version at Runtime

- Use the Build constrains class within your app to build conditions that ensure the code that depends on higher API levels is executed only when those APIs are available on the system.

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb
    // or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Question?

1. What are the main challenges in developing Android app?
2. What are the resources you need to create in order to make Android app supporting different languages?
3. “In order to make an image file compatible with all screen sizes, it is necessary to create it in the best quality possible” Comment on this statement.

Question?

4. What is the purpose of the uses-sdk element?
5. Why is it appropriate to set the minimum SDK version to the lowest version and the target SDK version to the latest version?
6. “The Android system performs scaling and resizing to make your application work on different screens. So it is OK to stick to the default settings.” Comment on this statement.

Review Question

7. Mobile devices come in different sizes and configuration. As a mobile app developer, explain TWO (2) techniques to ensure your app is compatible with devices that are of different screen sizes and language configuration.