ANDROID 6.0 RUNTIME PERMISSIONS

(A CODE LAB)

CODE LAB OBJECTIVE

EXPERIMENT WITH ANDROID 6.0 RUNTIME PERMISSIONS

- You do the experimenting!
- Or, let the presenter do the experimenting, while you sit back and relax, as why should *you* do all the work?

RUNTIME PERMISSIONS

LEGACY APPS

(`targetSdkVersion` < 23)

- No code changes
- Behavior akin to "app ops"
  - User can revoke `dangerous` permissions at runtime
  - Affected APIs return bogus results

RUNTIME PERMISSIONS

MODERN APPS

(`targetSdkVersion` >= 23)

- Same `uses-permission` elements
- Must request `dangerous` permissions
  - Modal dialog-style UI
  - User can accept, deny, or deny with extreme prejudice

RUNTIME PERMISSION MECHANICS

`checkSelfPermission()`

- `Context` and `ContextCompat`
- Given name of permission, tells you if you have it

RUNTIME PERMISSION MECHANICS

`requestPermissions()`

- `Activity` and `ActivityCompat`
- Given array of permission names, prompts user to accept/deny them
  - One "pane" to dialog per permission *group*
  - Get result in `onRequestPermissionsResult()`

RUNTIME PERMISSION MECHANICS

`shouldShowRequestPermissionRationale()`

- `Activity` and `ActivityCompat`
- Given permission name, returns `true` if...
  - ...you have never asked for this permission, or...
  - ...you asked, the user denied it, but the user has not blocked further requests
- Use: educate user about upcoming permission request

POSSIBLE RUNTIME PERMISSION STATES

- We have never asked for the permission
- We asked for the permission, and it was granted
- We asked for the permission, and it was denied
- We asked for the permission, and it was denied, and the user took out a restraining order against us

CODE LAB TIME!

CODE LAB RESOURCES

- Starter project
- PDF with instructions
- Finished project ...if you just want to see the results

TASK #0: INSTALL THE ANDROID 6.0 SDK

If you have done this already, great!
If you have not done this already... just sit back and watch!

TASK #1: IMPORT THE STARTER PROJECT

- `RuntimePermTutorial.zip` file
- Unzip in some likely spot
- File > New... > Import Project from Android Studio

REVIEWING THE SAMPLE APP

- Landscape and portrait layouts, two big buttons
  - Take a picture
  - Record a video
- Dependencies
  - Icon button library
  - CWAC-Cam2 for camera stuff

TASK #2: UPGRADE GRADLE FOR ANDROID 6.0

- `compileSdkVersion 23`
- `buildToolsVersion "23.0.0"`
- `targetSdkVersion 23`

SO, WHAT ARE WE GONNA DO ABOUT PERMISSIONS?

- Ask for `CAMERA` and `WRITE_EXTERNAL_STORAGE` on first run, as the app is totally useless without them
- Ask for `RECORD_AUDIO` when they click the "Record Video" button, as we will not need it before then
- Ask for whatever permissions we do not hold when they click a button that needs them
- If they deny permissions, then click a button, explain why we are going to ask for the permissions again

TASK #3: ADD FIELDS FOR FIRST-RUN DETECTION

```
private static final String PREF_IS_FIRST_RUN="firstRun";
private SharedPreferences prefs;
```

TASK #4: INITIALIZE THE PREFERENCES

Add the following to `onCreate()`:

```
prefs=PreferenceManager.getDefaultSharedPreferences(this);
```

TASK #5: USE THE PREFERENCES TO TRACK THE FIRST RUN

```
private boolean isFirstRun() {
  boolean result=prefs.getBoolean(PREF_IS_FIRST_RUN, true);

  if (result) {
    prefs.edit().putBoolean(PREF_IS_FIRST_RUN, false).apply();
  }

  return(result);
}
```

TASK #6: CHECK FOR FIRST RUN

Add the following to the bottom of `onCreate()`:

```
if (isFirstRun()) {
  // TODO
}
```

TASK #7: ADD SOME STATIC IMPORTS

```
import static android.Manifest.permission.CAMERA;
import static android.Manifest.permission.RECORD_AUDIO;
import static android.Manifest.permission.WRITE_EXTERNAL_STORAGE;
```

TASK #8: LIST OUR TAKE-PICTURE PERMISSIONS

```
private static final String[] PERMS_TAKE_PICTURE={
  CAMERA,
  WRITE_EXTERNAL_STORAGE
};
```

TASK #9: ADD OUR TAKE-PICTURE PERMISSION RESULT CODE

```
private static final int RESULT_PERMS_INITIAL=1339;
```

TASK #10: ADD THE SUPPORT LIBRARY FOR PERMISSION COMPATIBILITY CODE

```
dependencies {
  compile 'com.commonsware.cwac:cam2:0.2.+'
  compile 'com.githang:com-phillipcalvin-iconbutton:1.0.1@aar'
  compile 'com.android.support:support-v4:23.0.1'
}
```

TASK #11: ASK FOR PERMISSION

```
if (isFirstRun()) {
  ActivityCompat.requestPermissions(this, PERMS_TAKE_PICTURE,
    RESULT_PERMS_INITIAL);
}
```

TASK #12: ADD PERMISSION CALLBACK STUB

```java
@Override
public void onRequestPermissionsResult(int requestCode,
  String[] permissions, int[] grantResults) {
    // TODO
}
```

TASK #13: TRY IT OUT!

- Run the app ...and it should prompt you for permissions
- Press BACK
- Run the app again ...and it should **not** prompt you for permissions
- Uninstall the app ...so we start from scratch with permissions on the next run

TASK #14: CREATE A PERMISSION-CHECK HELPER METHOD

```
private boolean hasPermission(String perm) {
  return(ContextCompat.checkSelfPermission(this, perm)==
    PackageManager.PERMISSION_GRANTED);
}
```

TASK #15: SEE IF WE CAN TAKE A PICTURE

```
private boolean canTakePicture() {
  return(hasPermission(CAMERA) &&
    hasPermission(WRITE_EXTERNAL_STORAGE));
}
```

TASK #16: NO, I MEAN SEE IF WE CAN TAKE A PICTURE

```
public void takePicture(View v) {
  if (canTakePicture()) {
    takePictureForRealz();
  }
}
```

TASK #17: SEE IF WE SHOULD SHOW SOME RATIONALE

```
private boolean shouldShowTakePictureRationale() {
  return(ActivityCompat.shouldShowRequestPermissionRationale(this,
            CAMERA) ||
         ActivityCompat.shouldShowRequestPermissionRationale(this,
            WRITE_EXTERNAL_STORAGE));
}
```

TASK #18: USE THAT NEW METHOD, AS IT IS LONELY

```
public void takePicture(View v) {
  if (canTakePicture()) {
    takePictureForRealz();
  }
  else if (shouldShowTakePictureRationale()) {
    // TODO
  }
}
```

TASK #19: ADD A TEXTVIEW AS OUR "BREADCRUST"

- `@+id/breadcrust`
- `visibility` set to `gone`
- Add to both `layout` and `layout-land`

TASK #20: FIND OUR BREADCRUST

- `private TextView breadcrust;` as field
- `breadcrust= (TextView)findViewById(R.id.breadcrust);` in `onCreate()`

TASK #21: DEFINE A PICTURE RATIONALE MESSAGE

```
<string name="msg_take_picture">You need to grant us
permission! Tap the Take Picture button again, and we will ask
for permission.</string>
```

TASK #22: DEFINE ANOTHER RESULT CODE

```
private static final int RESULT_PERMS_TAKE_PICTURE=1340;
```

TASK #23: NET THE PERMISSIONS

- `requestPermissions()` prompts user for everything we ask for ...even if they granted the permission to us before
- This is an icky method, too big for this slide

TASK #24: SHOW RATIONALE WHEN NEEDED

"What is it that you want?"
"I want the code!"
"You can't handle the code!"
(...or at least this slide can't)

TASK #25: DEAL WITH THE RESULTS

- If we requested permissions, and we can take a picture, go ahead
- If we requested permissions, cannot take a picture, but should show rationale, do that
- Otherwise, we're stuck
- (and, yes, the code is too long for the slide here too)

TASK #26: TRY IT OUT!

- Run the app, reject one of the permissions
- Tap the picture button, get rationale
- Tap the picture button again, reject the permission again
- Uninstall the app

TASK #27: ONCE MORE, FROM THE TOP, WITH VIDEO

```
private boolean canRecordVideo() {
  return(canTakePicture() && hasPermission(RECORD_AUDIO));
}
```

TASK #28: ONLY RECORD IF WE CAN

```java
public void recordVideo(View v) {
  if (canRecordVideo()) {
    recordVideoForRealz();
  }
}
```

TASK #29: U CAN NEEDZ VIDEO RATIONALE?

```
private boolean shouldShowRecordVideoRationale() {
  return(shouldShowTakePictureRationale() ||
    ActivityCompat.shouldShowRequestPermissionRationale(this,
      RECORD_AUDIO));
}
```

TASK #30: ASK ALL THE PERMISSIONS! AND, UM, RESULTS TOO!

```
private static final String[] PERMS_ALL={
  CAMERA,
  WRITE_EXTERNAL_STORAGE,
  RECORD_AUDIO
};
private static final int RESULT_PERMS_RECORD_VIDEO=1341;
```

TASK #31: REALLY RECORD THE VIDEO. REALLY.

(pretend that there is some code here)

TASK #32: HANDLE THE RESULTS

(did I mention that runtime permissions are tedious?)

TASK #33: CONFIGURATION CHANGES. UGH.

```
private static final String STATE_BREADCRUST=
  "com.commonsware.android.perm.tutorial.breadcrust";

@Override
protected void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);

  if (breadcrust.getVisibility()==View.VISIBLE) {
    outState.putCharSequence(STATE_BREADCRUST,
      breadcrust.getText());
  }
}
```

TASK #33½: CONFIGURATION CHANGES. UGH.

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);

  CharSequence cs=savedInstanceState.getCharSequence(STATE_BREADCRUS

  if (cs!=null) {
    breadcrust.setVisibility(View.VISIBLE);
    breadcrust.setText(cs);
  }
}
```