IT332: Mobile Application Development

Lecture # 09 : Android Permissions - II

Muhammad Imran

Outline

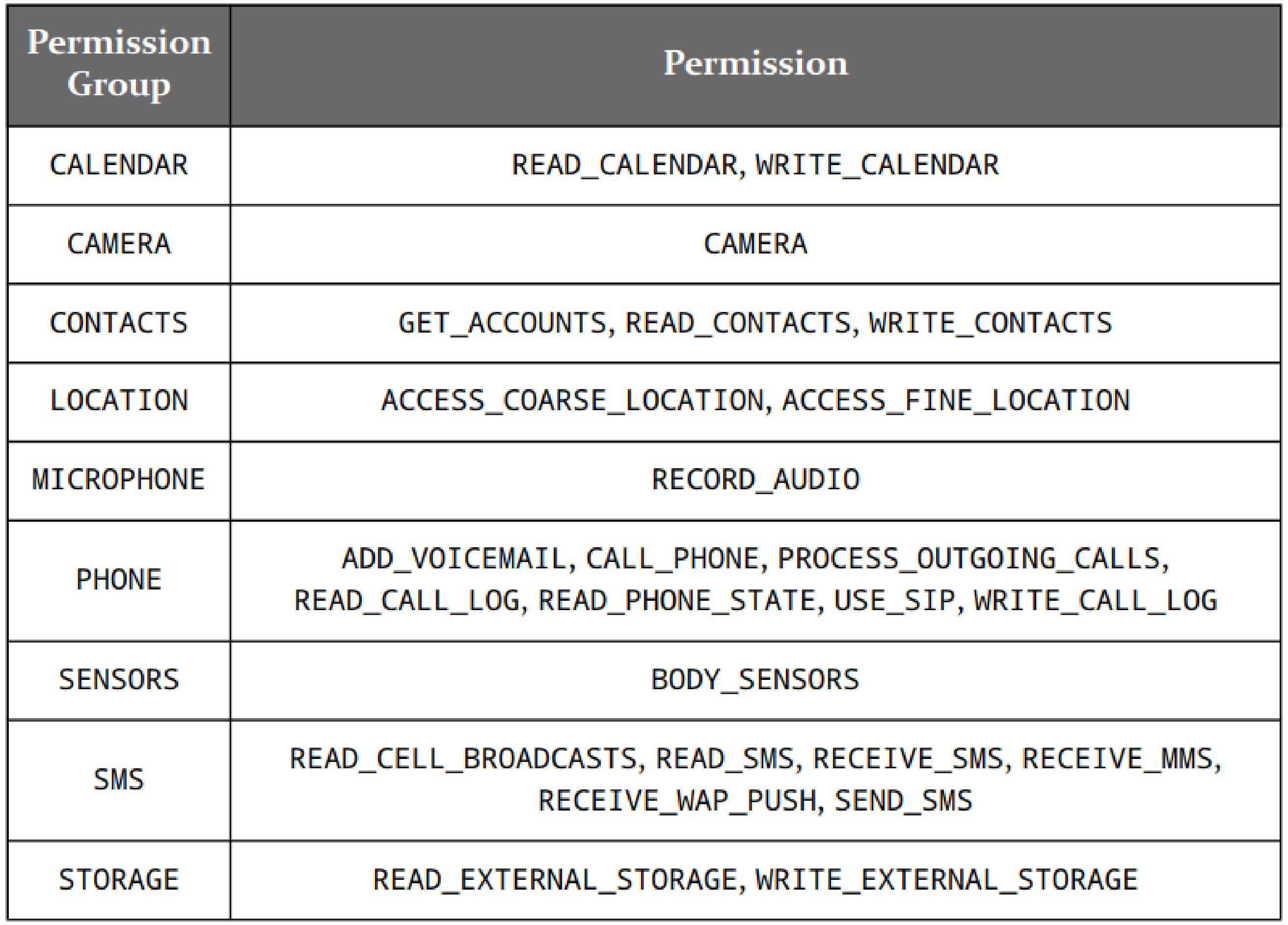
- Android 6.0+ Runtime Permission System
- Declare App Permissions
- Hardware Associated Permissions
- Requesting App Permissions
- Handle Permission Denial
- Compatibility
- Using Command Line

Android 6.0+ Runtime Permission System

- In Android 6.0 and higher devices, permissions that are dangerous not only have to be requested via <uses-permission> elements, but you also must ask the user to grant you those permissions at runtime.
- Users are not bothered with these permissions at install time, and you can delay asking until the user actually does something that needs them.

Android 6.0+ Runtime Permission System

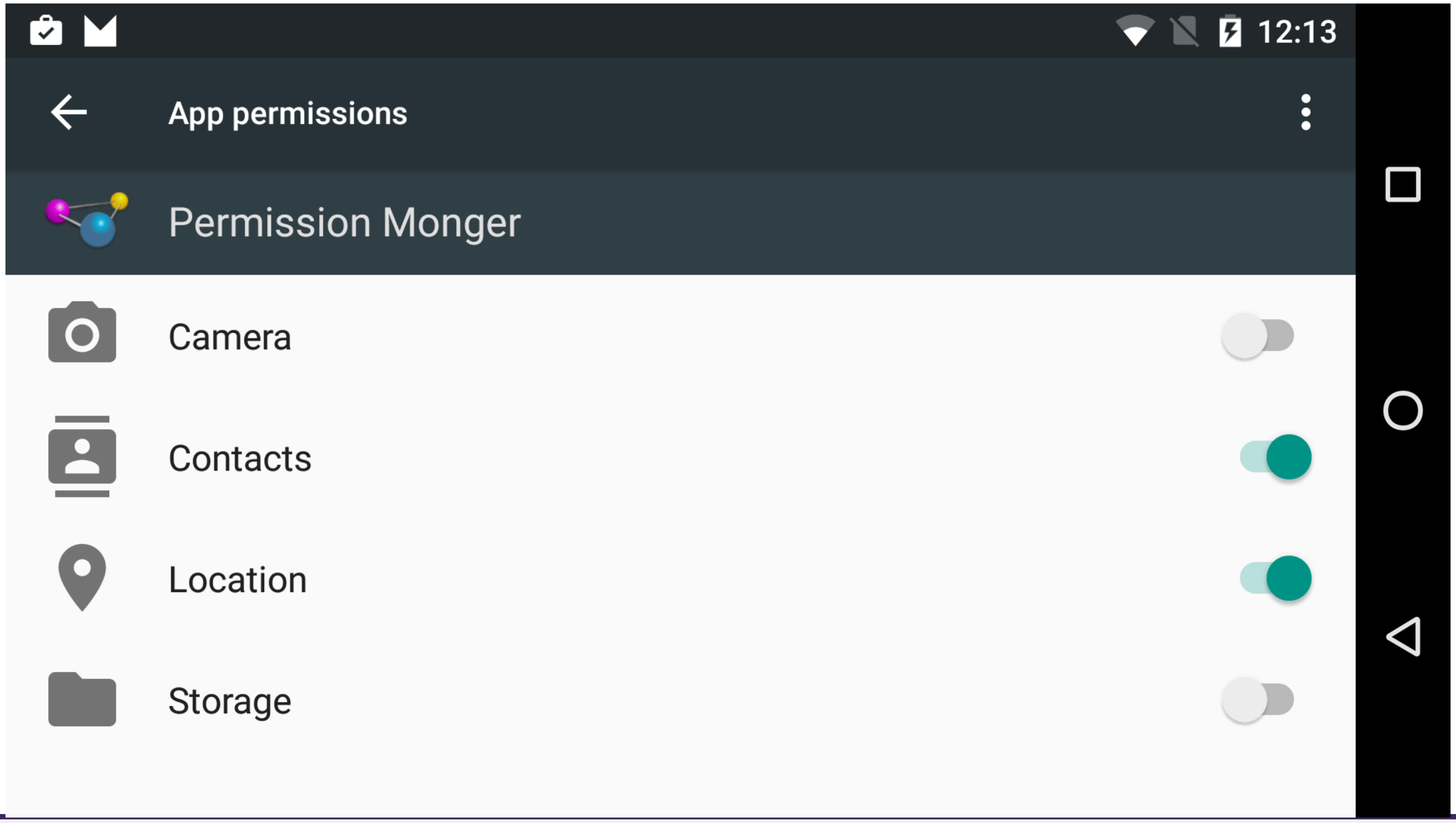
- There are nine permission groups that Android 6.0 manages as user-controllable permissions:



Permission Group	Permission
CALENDAR	READ_CALENDAR, WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	GET_ACCOUNTS, READ_CONTACTS, WRITE_CONTACTS
LOCATION	ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	ADD_VOICEMAIL, CALL_PHONE, PROCESS_OUTGOING_CALLS, READ_CALL_LOG, READ_PHONE_STATE, USE_SIP, WRITE_CALL_LOG
SENSORS	BODY_SENSORS
SMS	READ_CELL_BROADCASTS, READ_SMS, RECEIVE_SMS, RECEIVE_MMS, RECEIVE_WAP_PUSH, SEND_SMS
STORAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE

Android 6.0+ Runtime Permission System

- Users will be able to revoke permissions by group, through the Settings app.
- They can go into the page for your app, click on Permissions, and see a list of the permission groups for which you are requesting permissions:


12:13

App permissions

Permission Monger

Camera

Contacts

Location

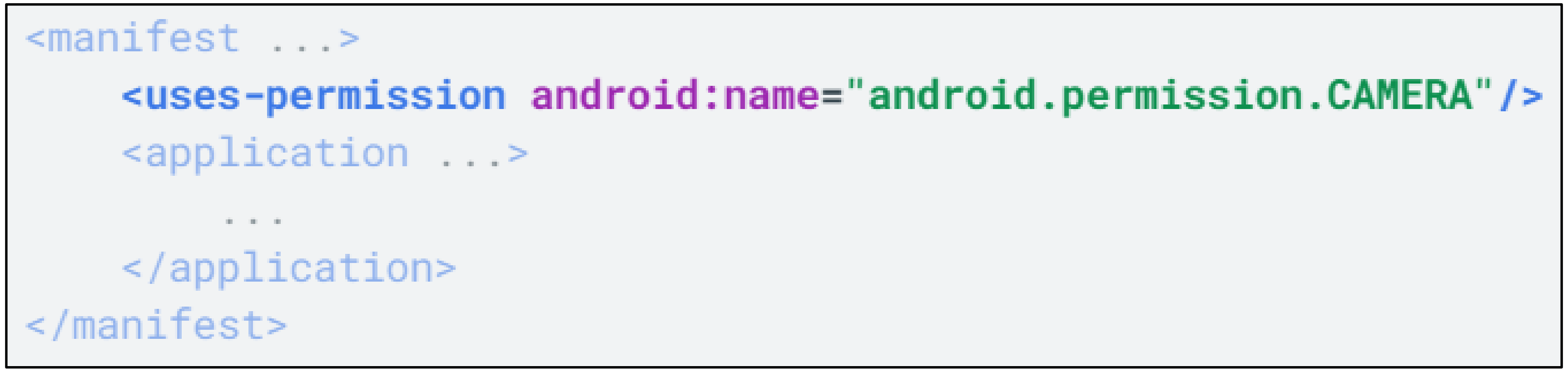
Storage

Declare App Permissions

- If your app requests app permissions, you must declare these permissions in your app's manifest file.
- These declarations help app stores and users understand the set of permissions that your app might request.

Add Declaration to App Manifest

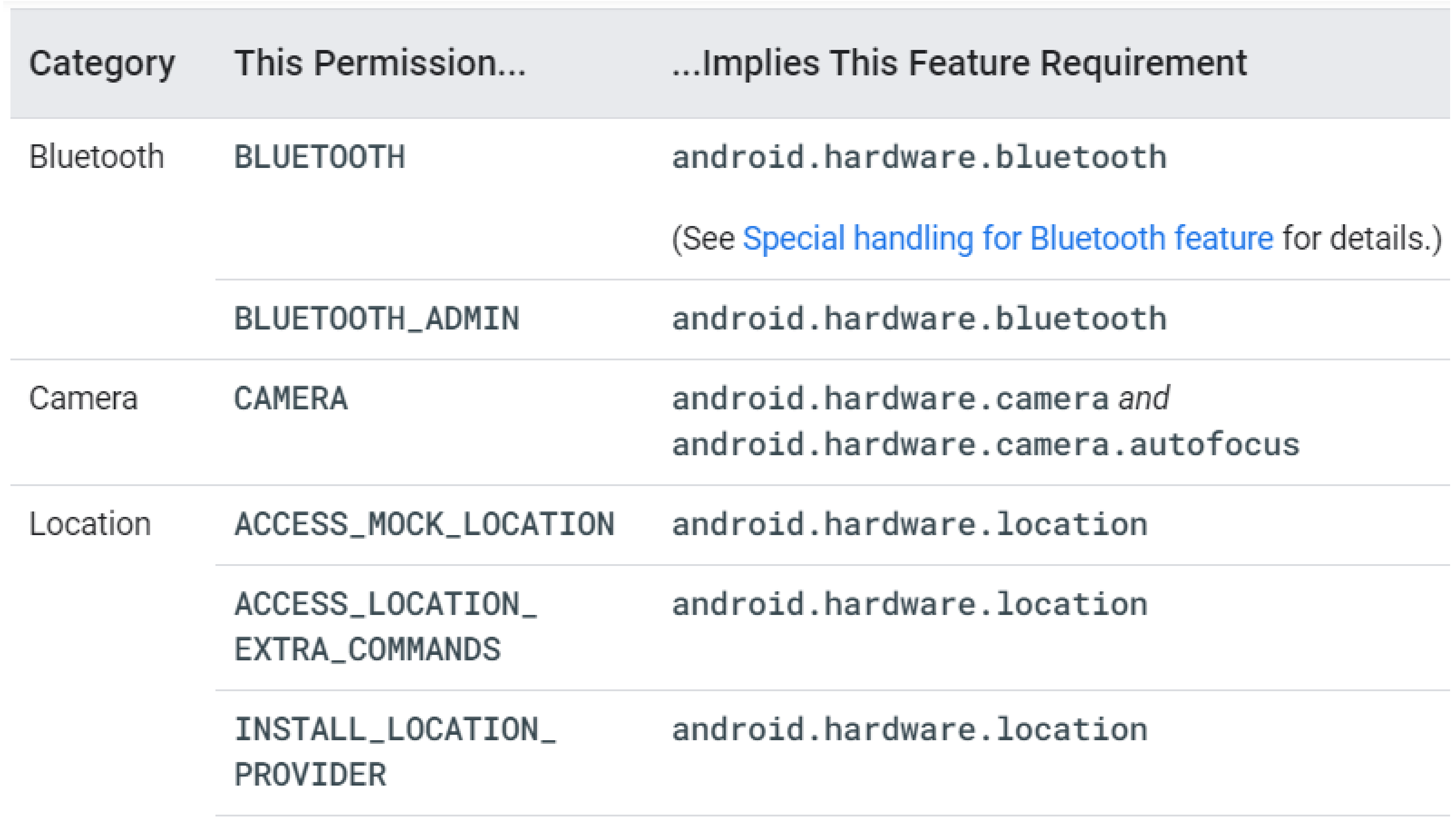
- To declare a permission that your app might request, include the appropriate <uses-permission> element in your app's manifest file.
- For example, an app that needs to access the camera would have this line in the manifest:



```
<manifest ...>
    <uses-permission android:name="android.permission.CAMERA"/>
    <application ...>
        ...
    </application>
</manifest>
```

Hardware-associated Permissions

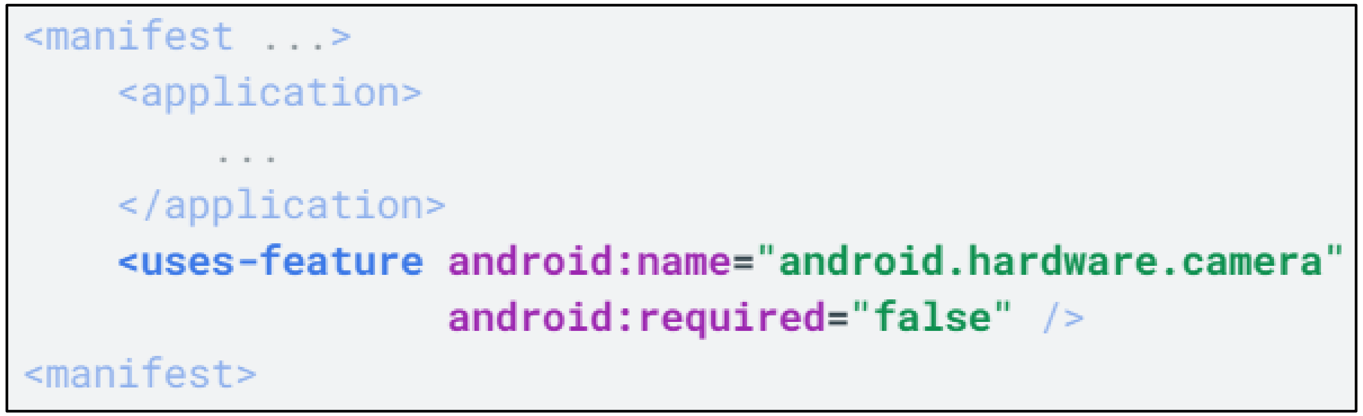
- You can find a complete list of hardware-associated permissions in the official documentation.



Category	This Permission...	...Implies This Feature Requirement
Bluetooth	BLUETOOTH	android.hardware.bluetooth (See Special handling for Bluetooth feature for details.)
	BLUETOOTH_ADMIN	android.hardware.bluetooth
Camera	CAMERA	android.hardware.camera *and* android.hardware.camera.autofocus
Location	ACCESS_MOCK_LOCATION	android.hardware.location
	ACCESS_LOCATION_ EXTRA_COMMANDS	android.hardware.location
	INSTALL_LOCATION_ PROVIDER	android.hardware.location

Declare Hardware as Optional

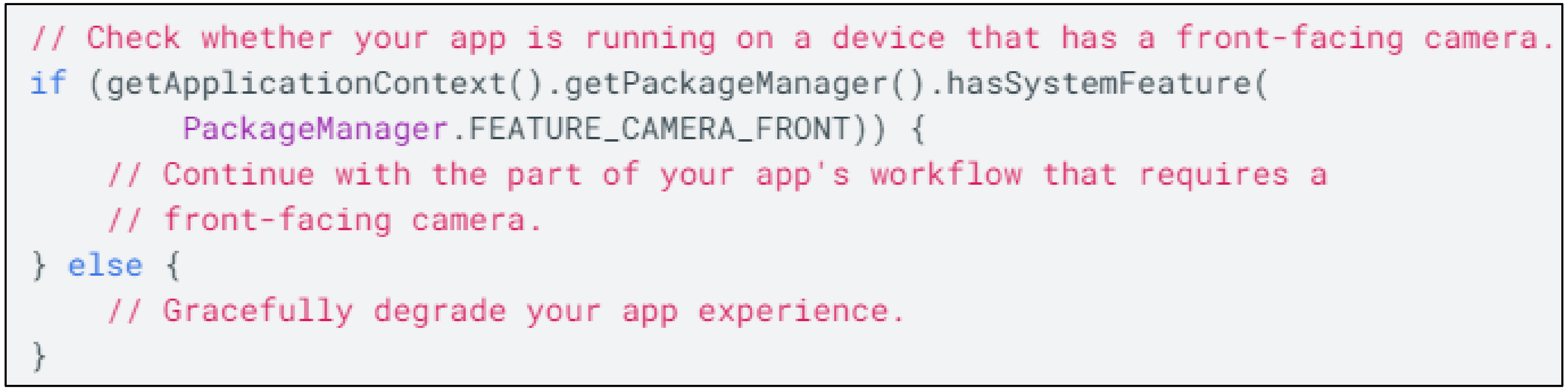
- Some permissions, such as CAMERA, allow your app to access pieces of hardware that only some Android devices have.
- If your app declares hardware-associated permissions, consider whether your app cannot run at all on a device that doesn't have that hardware.
- In most cases, hardware is optional, so it's better to declare the hardware as optional by setting android:required to false in your <uses-feature> declaration



```
<manifest ...>
    <application>
        ...
    </application>
    <uses-feature android:name="android.hardware.camera"
                  android:required="false" />
<manifest>
```

Determine Hardware Availability

- If you declare hardware as optional, it's possible for your app to run on a device that doesn't have that hardware.
- To check whether a device has a specific piece of hardware, use the `hasSystemFeature()` method.
- If the hardware isn't available, gracefully disable that feature in your app



```
// Check whether your app is running on a device that has a front-facing camera.
if (getApplicationContext().getPackageManager().hasSystemFeature(
        PackageManager.FEATURE_CAMERA_FRONT)) {
    // Continue with the part of your app's workflow that requires a
    // front-facing camera.
} else {
    // Gracefully degrade your app experience.
}
```

Declare Permissions by API Level

- To declare a permission only on devices that support runtime permissions—that is, devices that run Android 6.0 (API level 23) or higher—include the uses-permission-sdk-23 element instead of the uses-permission element.
- When using either of these elements, you can set the maxSdkVersion attribute.
- This attribute indicates that devices running a higher version than maxSdkVersion don't need a particular permission.

Basic Principles for Requesting

- If your app needs to use resources or information outside of its own sandbox, it should declare and request permissions
- The basic principles for requesting permissions at runtime include:
  - Ask for permissions in context, when the user starts to interact with the feature that requires it.
  - Don't block the user. Always provide the option to cancel an educational UI flow related to permissions.
  - If the user denies or revokes a permission that a feature needs, gracefully degrade your app, possibly by disabling that feature only.
  - Don't assume any system behavior.

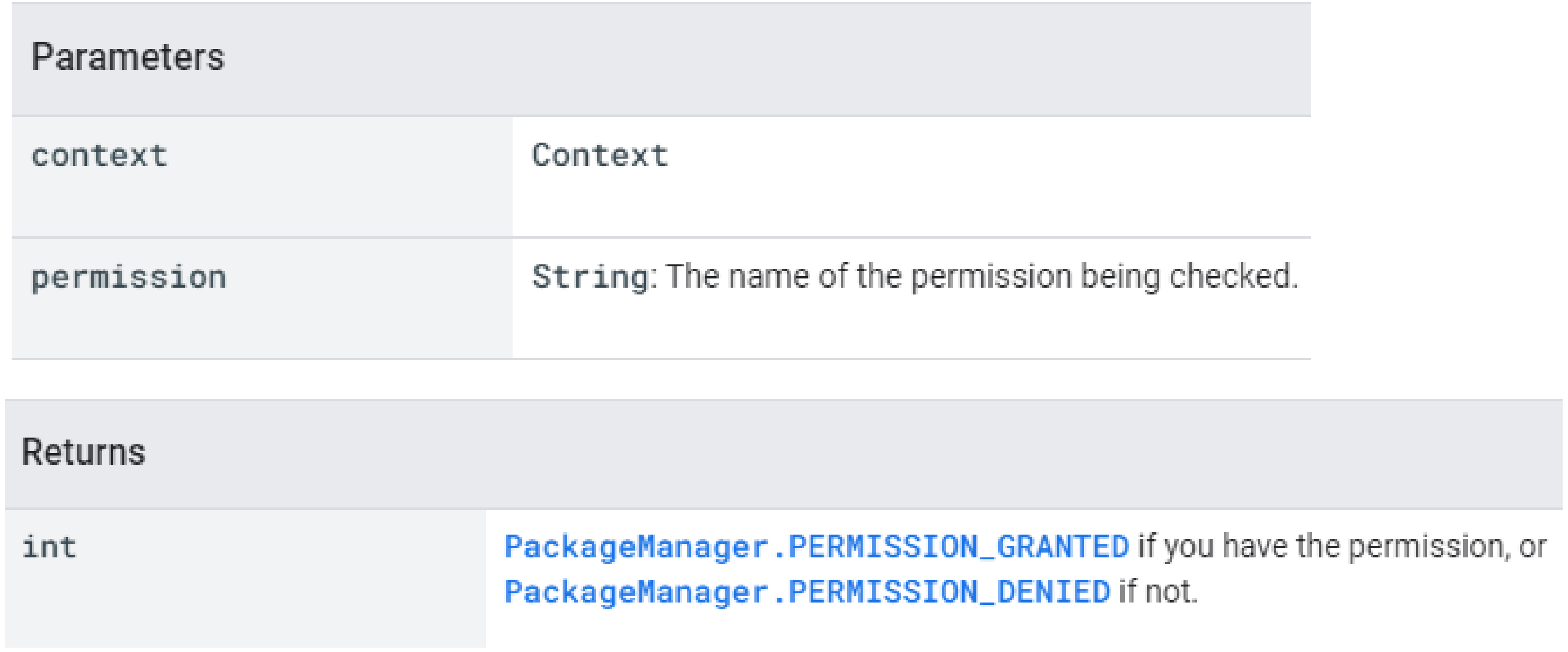
Requesting App Permissions

- Before requesting for a permission, it should be determined whether our app was already granted the permission
- To check if the user has already granted your app a particular permission, pass that permission into the ContextCompat.checkSelfPermission() method
- This method returns either PERMISSION_GRANTED or PERMISSION_DENIED, depending on whether your app has the permission.

ContextCompat.checkSelfPermission()

- ContextCompat.checkSelfPermission() method is used to determine whether you have been granted a particular permission.
- Method can be called by passing context and permission name

```
public static int checkSelfPermission (Context context, String permission)
```



Parameters	
context	Context
permission	String: The name of the permission being checked.

Returns	
int	PackageManager.PERMISSION_GRANTED if you have the permission, or PackageManager.PERMISSION_DENIED if not.

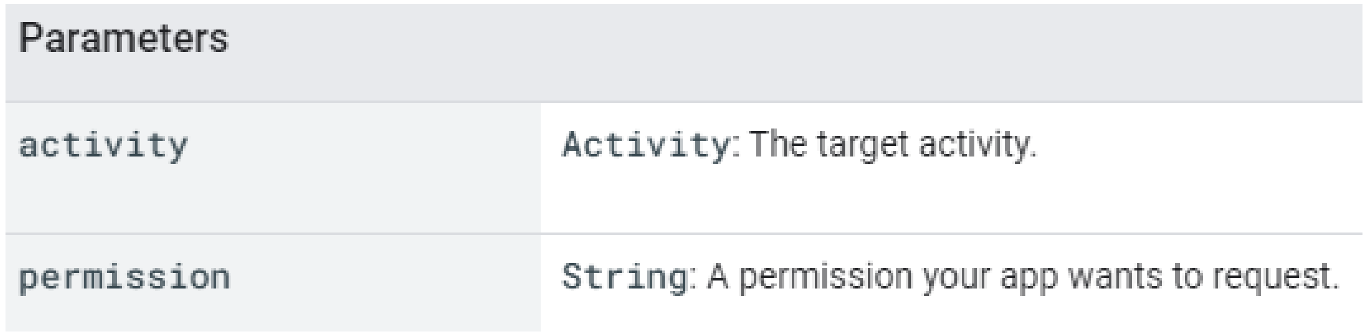
Explaining the Reason

- If user has once denied your request for a permission, you should explain him why you need a permission before requesting for the second time
- To check if you should show the permission rationale to the user, you can call shouldShowRequestPermissionRationale() method.
- If this method returns true, show an educational UI to the user.
- In this UI, describe why the feature, which the user wants to enable, needs a particular permission.

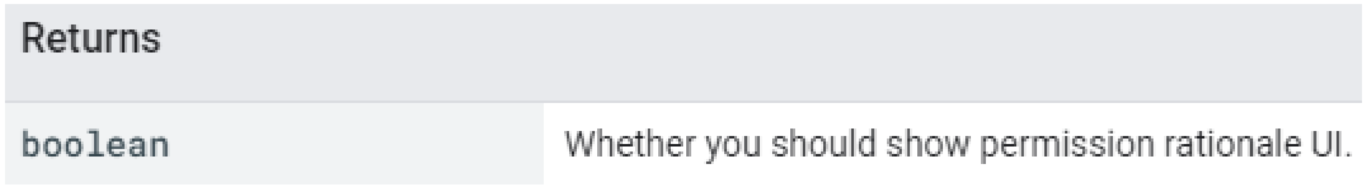
ActivityCompat.shouldShowRequestPermissionRationale()

- ActivityCompat.shouldShowRequestPermissionRationale() method gets whether you should show UI with rationale before requesting a permission..
- Method can be called by passing target activity and permission name

```
public static boolean shouldShowRequestPermissionRationale
                    (Activity activity, String permission)
```



Parameters	
activity	Activity: The target activity.
permission	String: A permission your app wants to request.



Returns	
boolean	Whether you should show permission rationale UI.

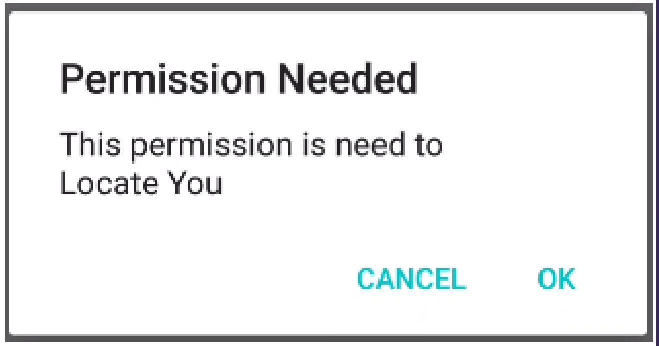
developers

Explaining the Reason

- If user has once denied your request for a permission, you should explain him why you need a permission before requesting for the second time
- To check if you should show the permission rationale to the user, you can call shouldShowRequestPermissionRationale() method.
- If this method returns true, show an educational UI to the user.
- In this UI, describe why the feature, which the user wants to enable, needs a particular permission.

Using Alert Dialog

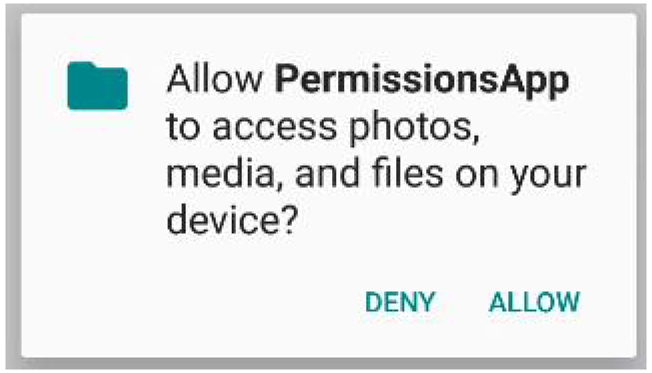
- You can create an alert dialog for showing the rationale to user.
- To create and show the dialog
  - `new AlertDialog.Builder(context).build().show()`
- To set other attributes of the dialog
  - `setTitle("Permission Needed")`
  - `setMessage("This permission is need to Locate You")`
  - `setPositiveButton("OK", new DialogInterface.OnClickListener()`
  - `setNegativeButton("Cancel", new DialogInterface.OnClickListener()`

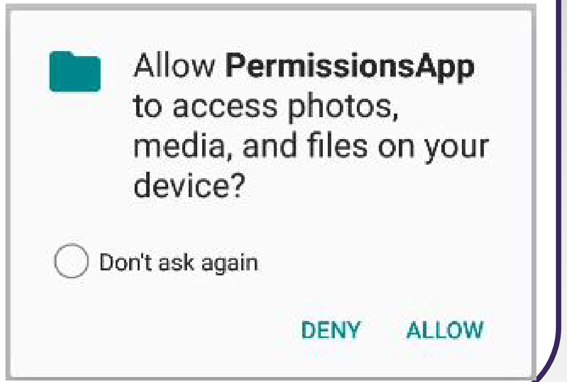

Permission Needed

This permission is need to Locate You

Request Permissions

- After the user views an educational UI, or the return value of shouldShowRequestPermissionRationale() is false, request the permission.
- Users see a system permission dialog, where they can choose whether to grant a particular permission to your app.
- To requests permissions to be granted to application, we can call requestPermissions() method
- If your app does not have the requested permissions the user will be presented with UI for accepting them

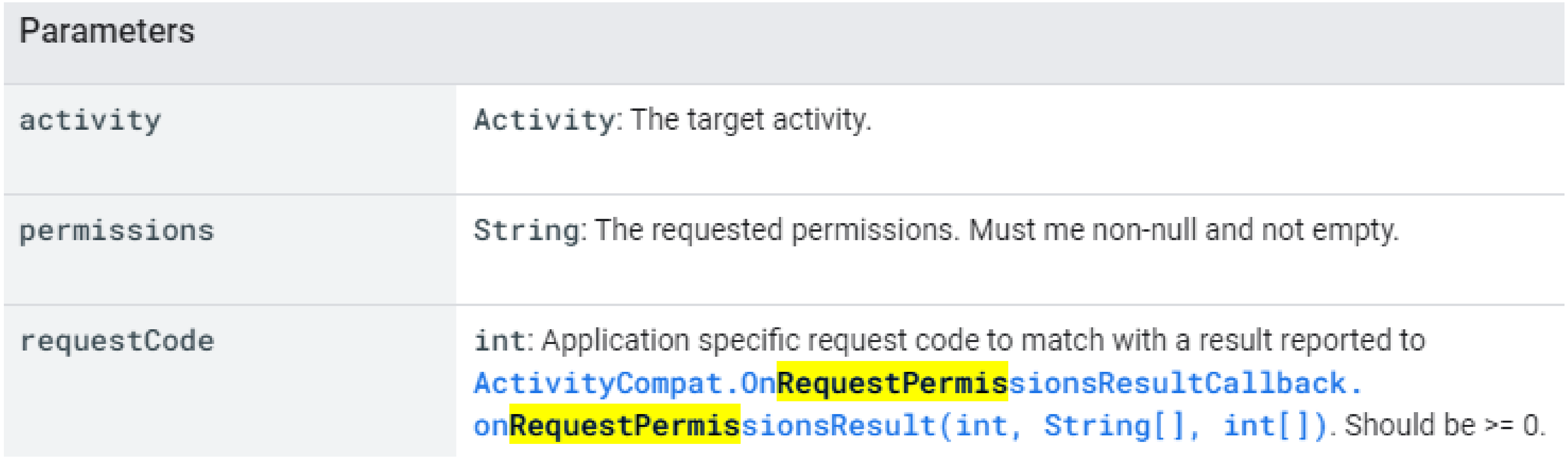

Allow **PermissionsApp** to access photos, media, and files on your device?


Allow **PermissionsApp** to access photos, media, and files on your device?

ActivityCompat.requestPermissions()

- This method can be called by passing activity , permission names and request code

```
public static void requestPermissions
        (Activity activity, String[] permissions, int requestCode)
```



Parameters	
`activity`	`Activity`: The target activity.
`permissions`	`String`: The requested permissions. Must me non-null and not empty.
`requestCode`	`int`: Application specific request code to match with a result reported to `ActivityCompat.OnRequestPermissionsResultCallback.onRequestPermissionsResult(int, String[], int[])`. Should be >= 0.

developers

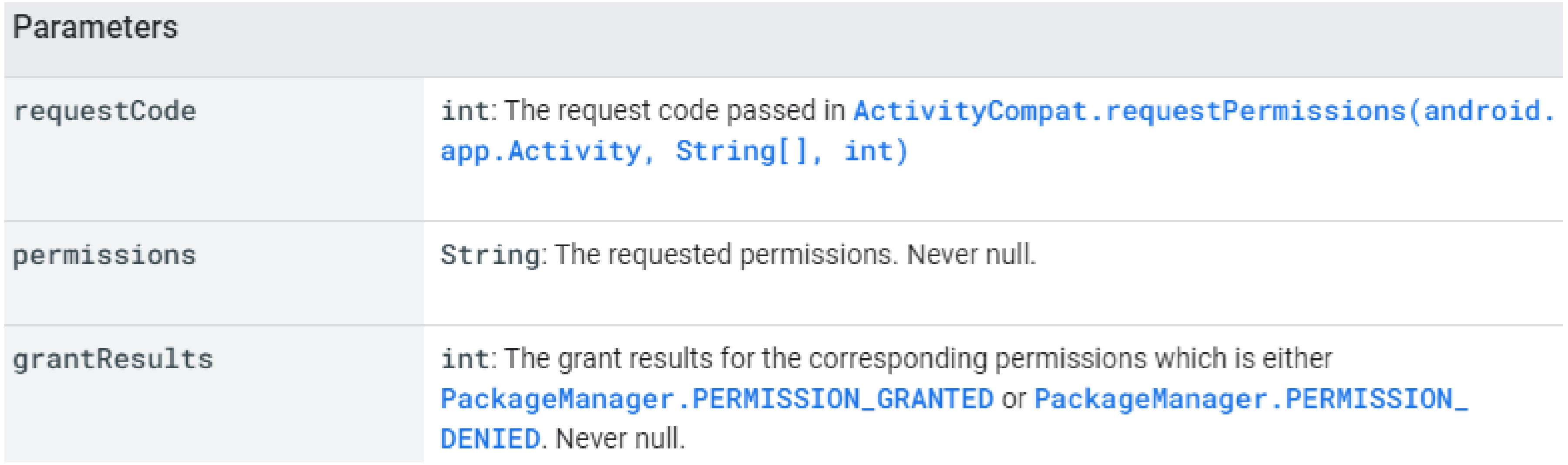
Permissions Request Result

- After the user has accepted or rejected the requested permissions you will receive a callback reporting whether the permissions were granted or not
- Your activity has to implement/override onRequestPermissionsResult() method
- This method is invoked for every call on ActivityCompat.requestPermissions()

ActivityCompat.onRequestPermissionsResult()

- This method is a callback method, hence you are only required to override it

```
public abstract void onRequestPermissionsResult
          (int requestCode, String[] permissions, int[] grantResults)
```



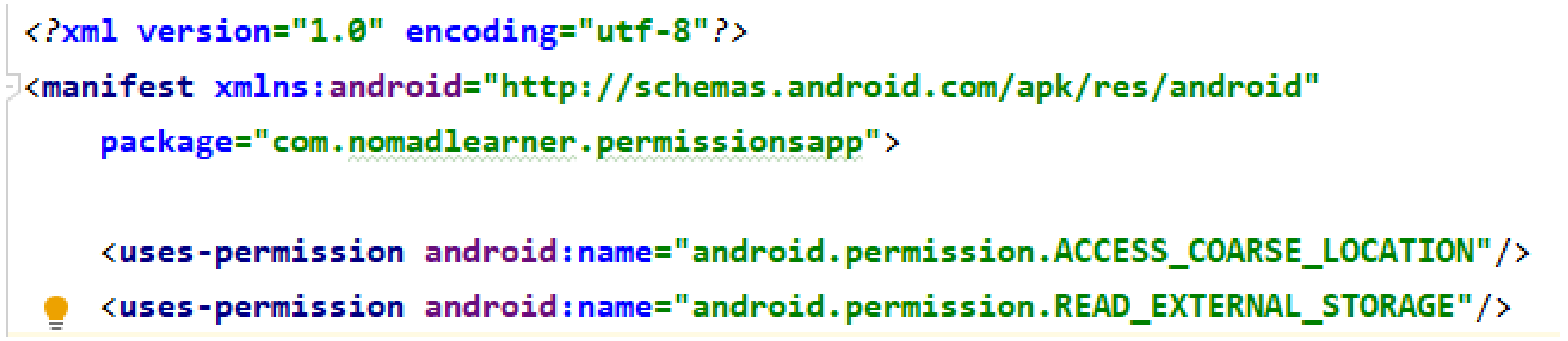
Parameters	
requestCode	int: The request code passed in ActivityCompat.requestPermissions(android.app.Activity, String[], int)
permissions	String: The requested permissions. Never null.
grantResults	int: The grant results for the corresponding permissions which is either PackageManager.PERMISSION_GRANTED or PackageManager.PERMISSION_DENIED. Never null.

developers

- It is possible that the permissions request interaction with the user is interrupted. In this case you will receive empty permissions and results arrays which should be treated as a cancellation.

Complete Flow

1. Declare permissions in the Manifest

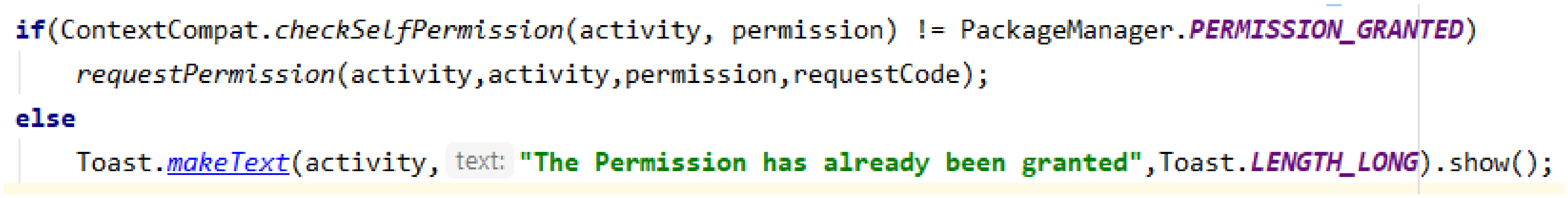


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.nomadlearner.permissionsapp">

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Complete Flow

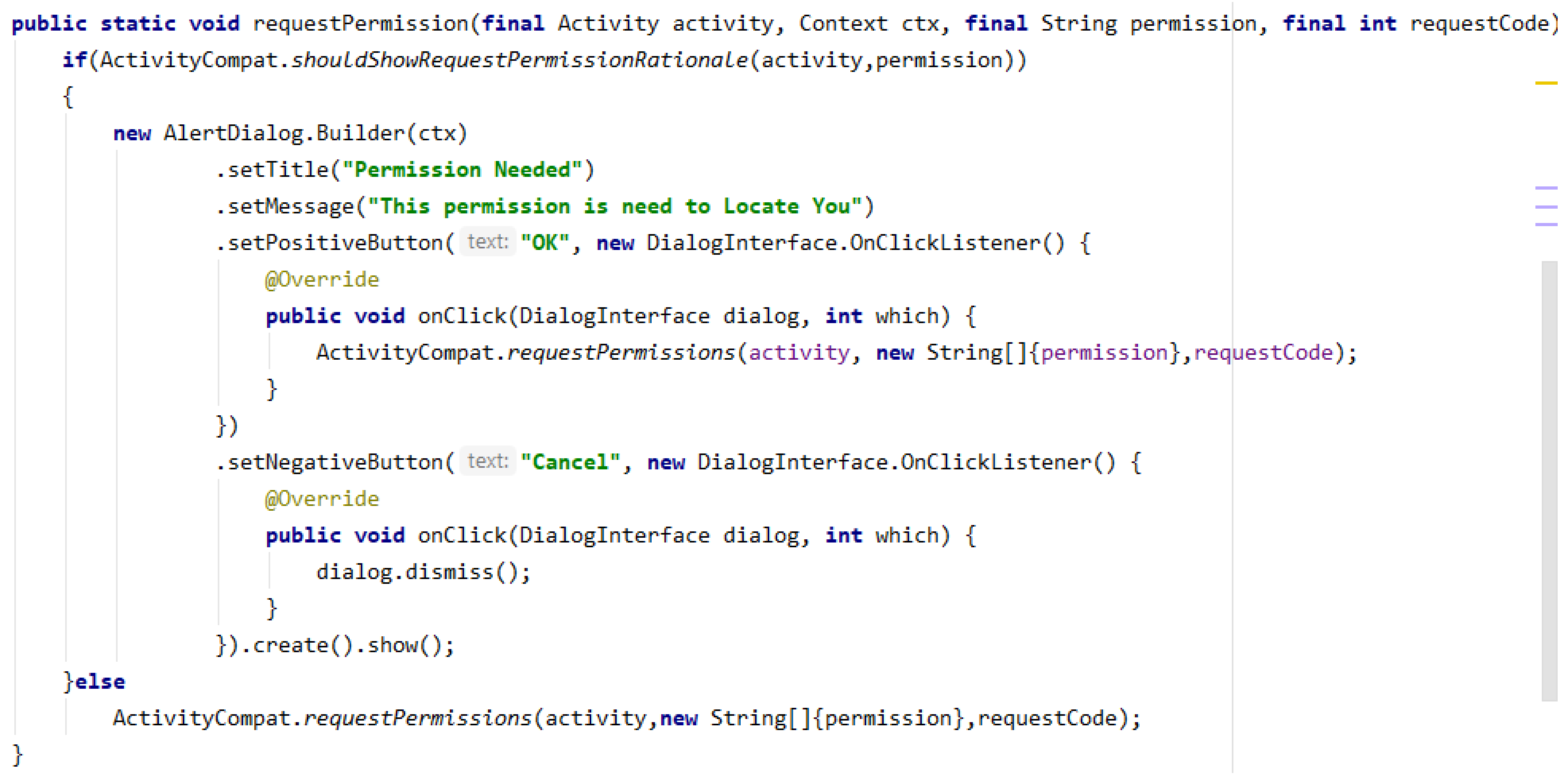
2. Check for the Permissions if Granted or Not



```
if(ContextCompat.checkSelfPermission(activity, permission) != PackageManager.PERMISSION_GRANTED)
    requestPermission(activity,activity,permission,requestCode);
else
    Toast.makeText(activity, text: "The Permission has already been granted",Toast.LENGTH_LONG).show();
```

Complete Flow

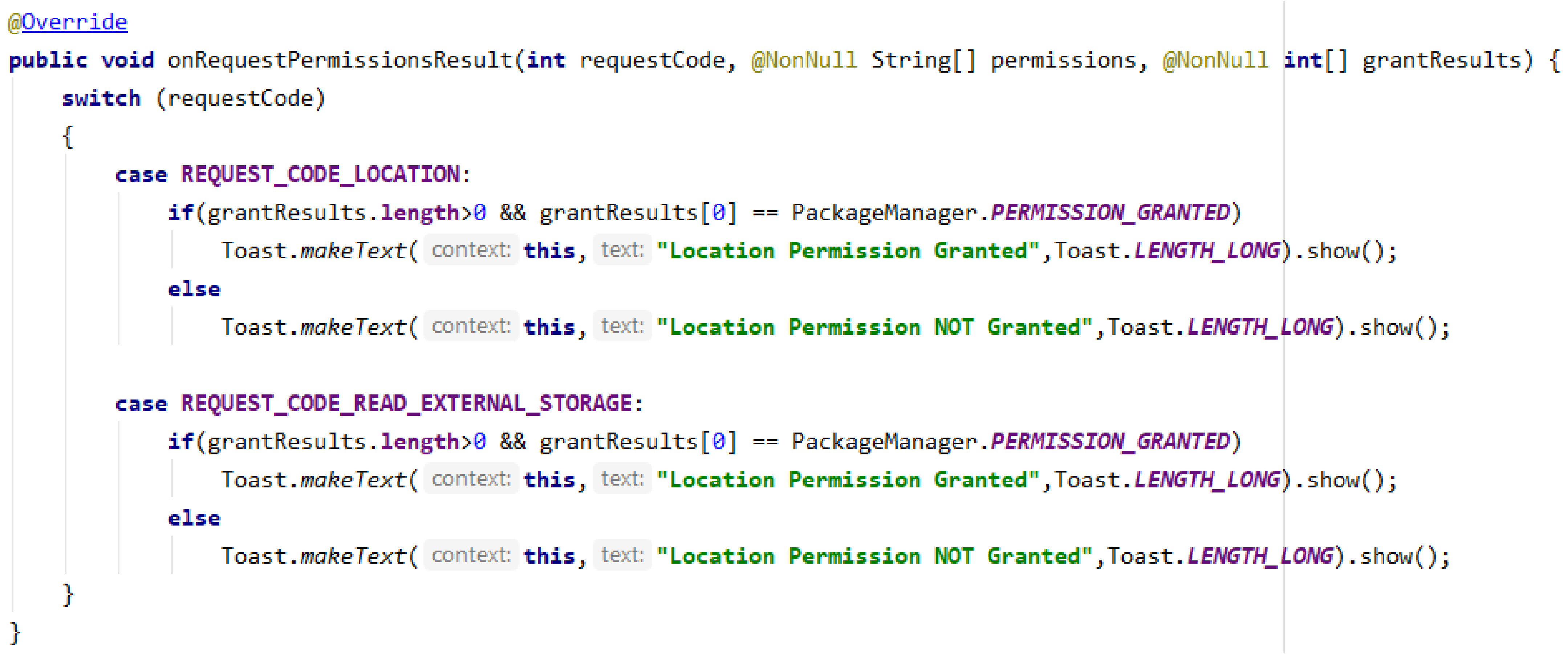
3. Request permission and show rationale if required



```
public static void requestPermission(final Activity activity, Context ctx, final String permission, final int requestCode)
    if(ActivityCompat.shouldShowRequestPermissionRationale(activity,permission))
    {
        new AlertDialog.Builder(ctx)
                .setTitle("Permission Needed")
                .setMessage("This permission is need to Locate You")
                .setPositiveButton( text: "OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        ActivityCompat.requestPermissions(activity, new String[]{permission},requestCode);
                    }
                })
                .setNegativeButton( text: "Cancel", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                }).create().show();
    }else
        ActivityCompat.requestPermissions(activity,new String[]{permission},requestCode);
}
```

Complete Flow

4. Handle the response

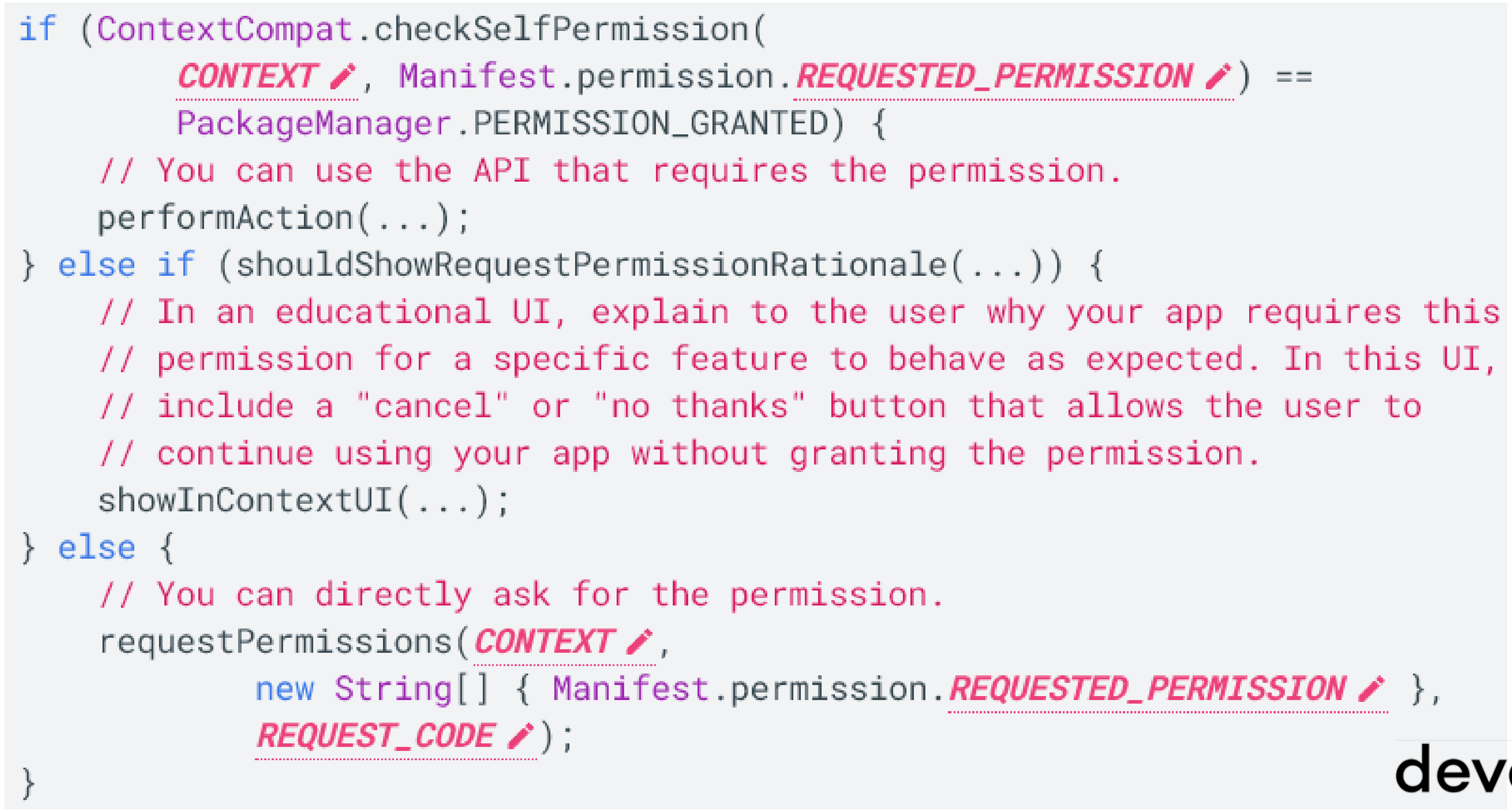


```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    switch (requestCode)
    {
        case REQUEST_CODE_LOCATION:
            if(grantResults.length>0 && grantResults[0] == PackageManager.PERMISSION_GRANTED)
                Toast.makeText( context: this, text: "Location Permission Granted",Toast.LENGTH_LONG).show();
            else
                Toast.makeText( context: this, text: "Location Permission NOT Granted",Toast.LENGTH_LONG).show();

        case REQUEST_CODE_READ_EXTERNAL_STORAGE:
            if(grantResults.length>0 && grantResults[0] == PackageManager.PERMISSION_GRANTED)
                Toast.makeText( context: this, text: "Location Permission Granted",Toast.LENGTH_LONG).show();
            else
                Toast.makeText( context: this, text: "Location Permission NOT Granted",Toast.LENGTH_LONG).show();
    }
}
```

Requesting App Permissions

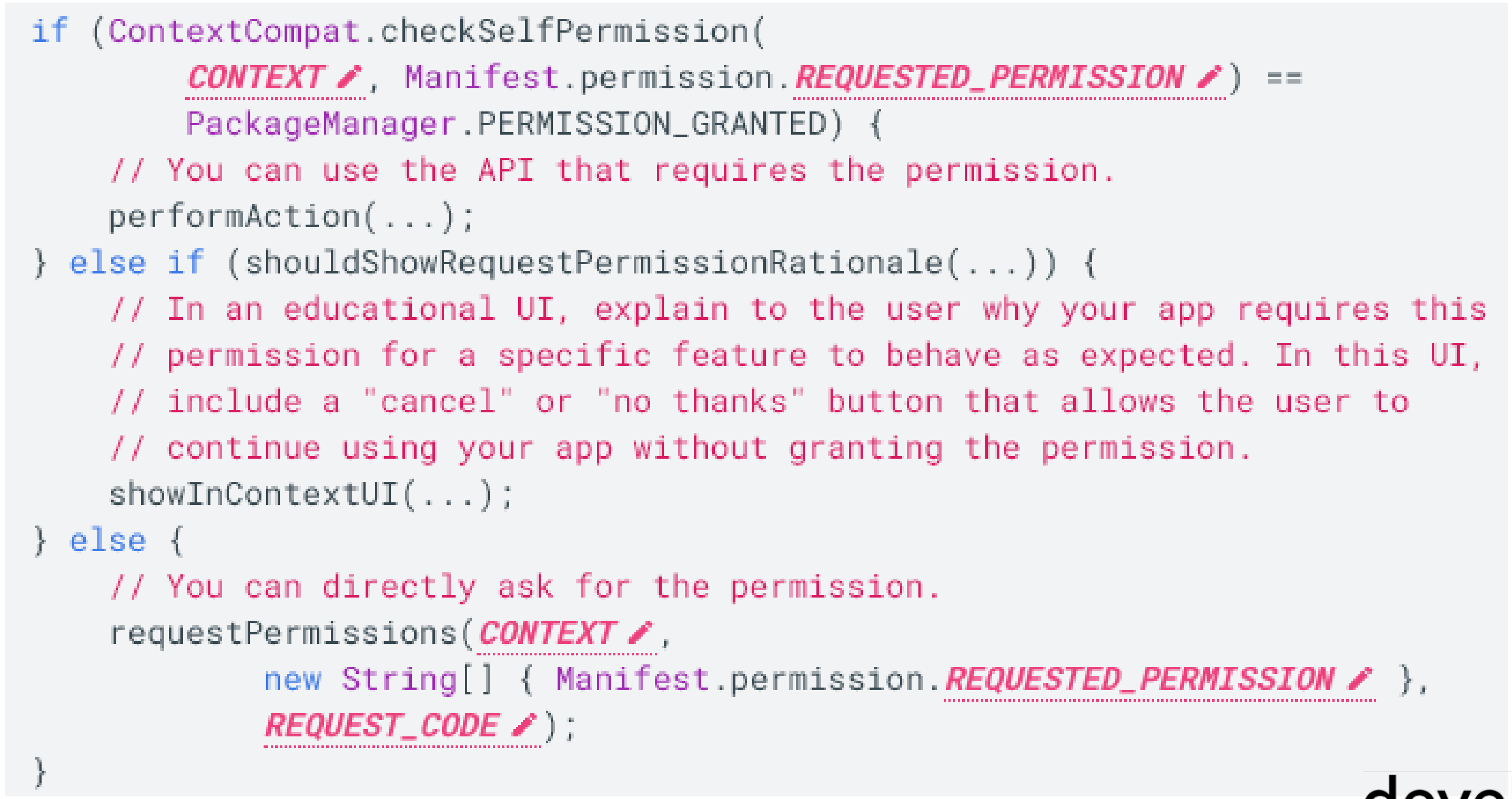
- The following code snippet demonstrates how to request a permission using a request code:



```
if (ContextCompat.checkSelfPermission(
        CONTEXT, Manifest.permission.REQUESTED_PERMISSION) ==
        PackageManager.PERMISSION_GRANTED) {
    // You can use the API that requires the permission.
    performAction(...);
} else if (shouldShowRequestPermissionRationale(...)) {
    // In an educational UI, explain to the user why your app requires this
    // permission for a specific feature to behave as expected. In this UI,
    // include a "cancel" or "no thanks" button that allows the user to
    // continue using your app without granting the permission.
    showInContextUI(...);
} else {
    // You can directly ask for the permission.
    requestPermissions(CONTEXT,
            new String[] { Manifest.permission.REQUESTED_PERMISSION },
            REQUEST_CODE);
}
```

Requesting App Permissions

- Handling the response



```
if (ContextCompat.checkSelfPermission(
        CONTEXT, Manifest.permission.REQUESTED_PERMISSION) ==
        PackageManager.PERMISSION_GRANTED) {
    // You can use the API that requires the permission.
    performAction(...);
} else if (shouldShowRequestPermissionRationale(...)) {
    // In an educational UI, explain to the user why your app requires this
    // permission for a specific feature to behave as expected. In this UI,
    // include a "cancel" or "no thanks" button that allows the user to
    // continue using your app without granting the permission.
    showInContextUI(...);
} else {
    // You can directly ask for the permission.
    requestPermissions(CONTEXT,
            new String[] { Manifest.permission.REQUESTED_PERMISSION },
            REQUEST_CODE);
}
```

developers

Handle Permission Denial

- If the user denies a permission request, the app should help users understand the implications and make users aware of the features that don't work
- Following best practices can be adapted:
  - Guide the user's attention. Highlight a specific part of your app's UI where there's limited functionality
  - Be specific. Don't display a generic message; instead, mention which features are unavailable because your app doesn't have the necessary permission.
  - Don't block the user interface. In other words, don't display a full-screen warning message that prevents users from continuing to use your app at all.

A Possible Code Tweak

- For a simpler boolean check to see if you have the permission, you could have your own hasPermission() method:

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

- Then you can use that hasPermission() call where you need it.

```
private void updateTable() {
  location.setText(String.valueOf(canAccessLocation()));
  camera.setText(String.valueOf(canAccessCamera()));
  internet.setText(String.valueOf(hasPermission(Manifest.permission.INTERNET)));
  contacts.setText(String.valueOf(canAccessContacts()));
  storage.setText(String.valueOf(hasPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE)));
}
```

A Possible Code Tweak

```
private boolean canAccessLocation() {
  return(hasPermission(Manifest.permission.ACCESS_FINE_LOCATION));
}

private boolean canAccessCamera() {
  return(hasPermission(Manifest.permission.CAMERA));
}

private boolean canAccessContacts() {
  return(hasPermission(Manifest.permission.READ_CONTACTS));
}
```

A Possible Code Tweak

- INITIAL_PERMS and INITIAL_REQUEST are just static final data members:

```java
private static final String[] INITIAL_PERMS={
  Manifest.permission.ACCESS_FINE_LOCATION,
  Manifest.permission.READ_CONTACTS
};
private static final int INITIAL_REQUEST=1337;
```

- In onCreate() to see if we can access locations or access contacts, and if not, it will request access to those two

```java
if (!canAccessLocation() || !canAccessContacts()) {
  requestPermissions(INITIAL_PERMS, INITIAL_REQUEST);
}
```

Compatibility

- The checkSelfPermission() method on Context is only available on API Level 23, thus you can add a check of the API level of the device you are running on:

```
if (Build.VERSION.SDK_INT>=Build.VERSION_CODES.M) {
  if (checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE)==
    PackageManager.PERMISSION_GRANTED) {
    // do something cool
  }
}
```

- A simpler approach is to use ContextCompat, from the support-v4 library.
- This has a static implementation of checkSelfPermission() that takes a Context and your permission string as parameters.

What Happens if the User Clears My App’s Data?

- If the user clears your app’s data through the Settings app, the runtime permissions are cleared as well.
- Behavior at this point will be as if your app had been just installed — checkSelfPermission() will return PERMISSION_DENIED, and you will need to request the permissions.

Using the Command Line

- For testing and debugging purposes, there are some command-line options for granting and revoking permissions that you can use.
- You can manually grant permissions via the adb shell pm grant command.
- This takes the application ID of your app and the fully-qualified name of the permission:

```
adb shell pm grant com.commonsware.android.perm.tutorial android.permission.CAMERA
```

- Similarly, you can use adb shell pm revoke to revoke a permission that was already granted to the app:

```
adb shell pm revoke com.commonsware.android.perm.tutorial android.permission.CAMERA
```

Recommended Readings

- Page # 589 to 606, Chapter: Requesting Permissions from The Busy Coder's Guide to Android Development, Final Version by Mark L. Murphy, 2019

- User Guide: https://developer.android.com/guide/topics/permissions/overview