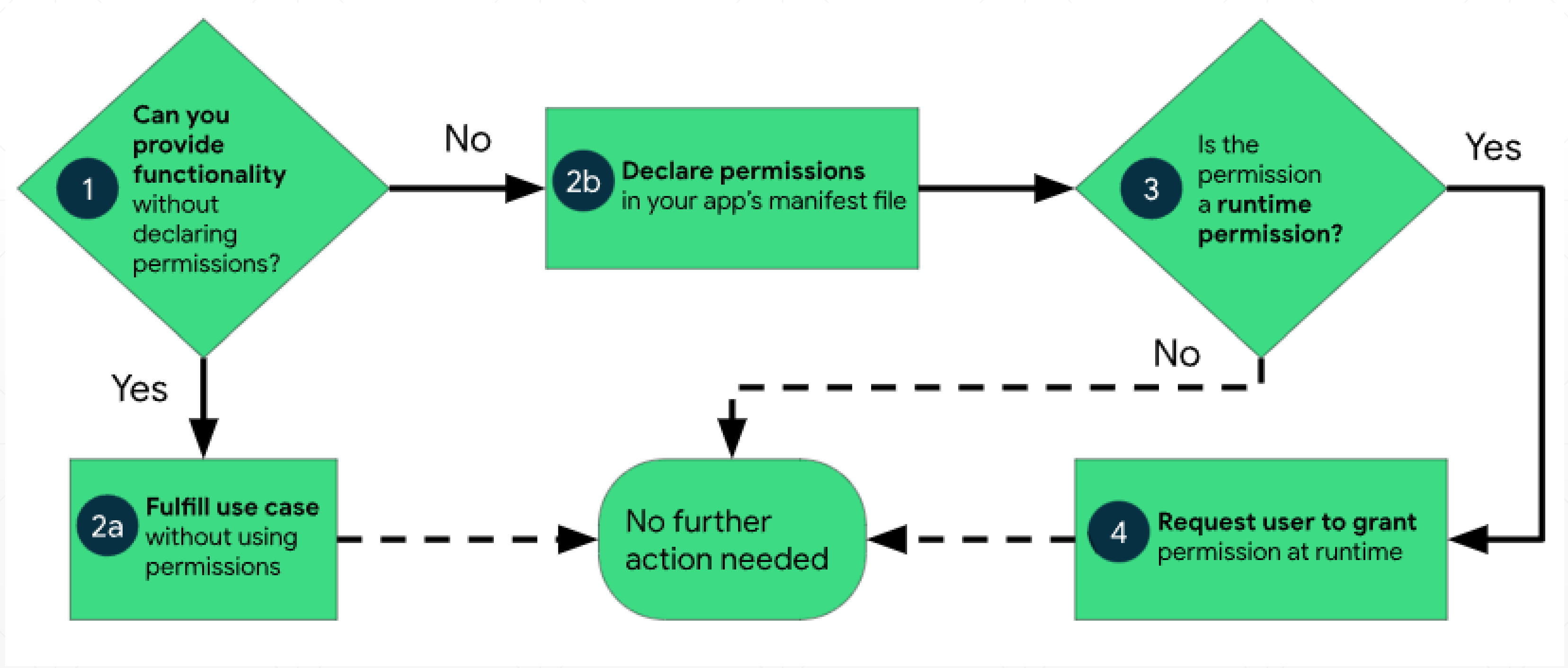
Mobile Programming

Permission

Permission (1/2)

- App permissions help support user privacy by protecting access to the
  - Restricted data, such as system state and a user's contact information
  - Restricted actions, such as connecting to a paired device and recording audio
- Permission workflow



1 Can you provide functionality without declaring permissions?
No
2b Declare permissions in your app's manifest file
3 Is the permission a runtime permission?
Yes
Yes
No
2a Fulfill use case without using permissions
No further action needed
4 Request user to grant permission at runtime

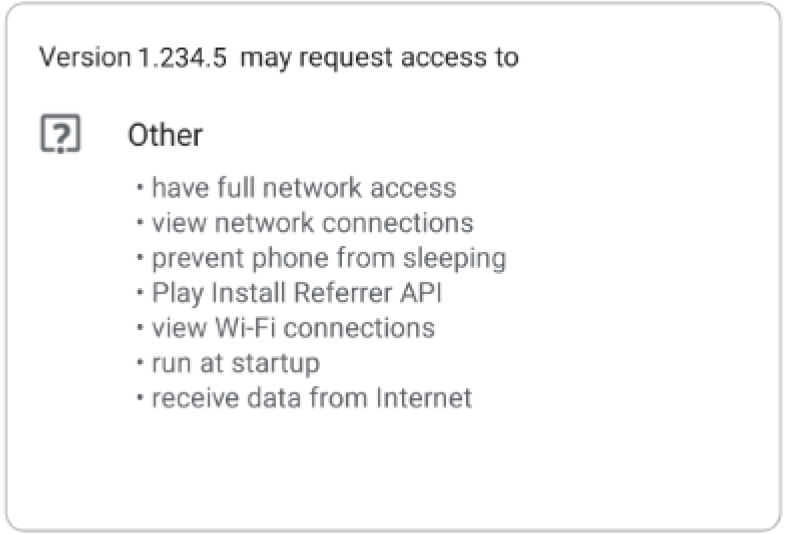
Permission (2/2)

- Provide functionality without needing to declare permissions?
  - When another installed app might be able to perform the functionality on your app’s behalf!
  - In these cases, you should delegate the task to another app using an intent
  - In doing so, you ***don’t need to declare the necessary permissions*** because the other app declares the permission instead

- Usecases
  - Take a photo / Record a video (use intent!)
  - https://developer.android.com/training/permissions/evaluating

Permission: Type (1/3)

- Otherwise… declare permissions!
  - Install-time permissions
    - Give your app limited access to restricted data, and they allow your app to perform restricted actions **that minimally affect** the system or other apps
  - An app store presents an install-time permission notice to the user when they view an app's details page


Version 1.234.5 may request access to

Other

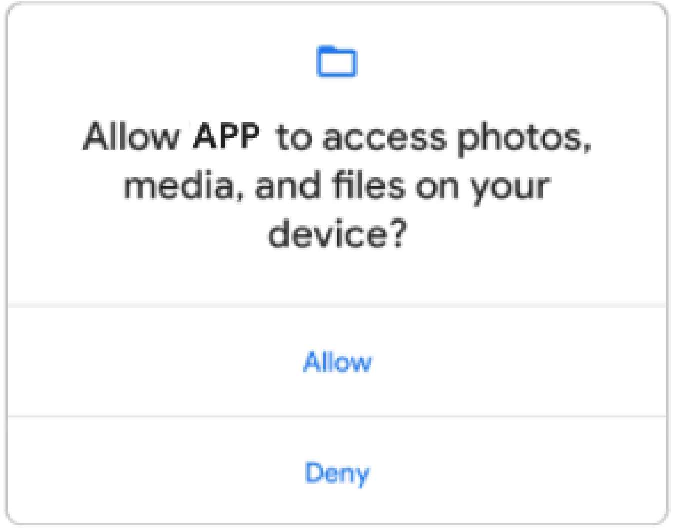
- have full network access
- view network connections
- prevent phone from sleeping
- Play Install Referrer API
- view Wi-Fi connections
- run at startup
- receive data from Internet

Permission: Type (2/3)

- Otherwise… declare permissions!
  - Install-time permissions
    - Normal permissions
      - Allow access to data and actions that extend beyond your app's sandbox
      - The data and actions **present very little risk** to the user's privacy, and the operation of other apps
      - The system assigns the "normal" protection level to normal permissions
    - Signature permissions
      - If the app declares a signature permission that another app has defined, and if the two apps are signed by the same certificate, then the system grants the permission to the first app at install time
      - The system assigns the "signature" protection level to signature permissions

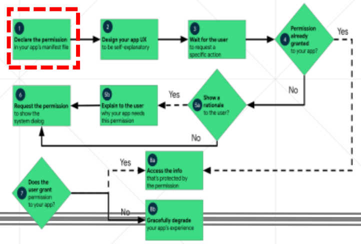
Permission: Type (3/3)

- Otherwise… declare permissions!
  - Runtime permissions (also known as dangerous permissions)
    - Give your app additional access to restricted data, and they allow your app to perform restricted actions **that more substantially affect** the system and other apps
    - You need to **request runtime permissions in your app** before you can access the restricted data or perform restricted actions
    - The system assigns the "dangerous" protection level to runtime permissions


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

- Permissions & protection levels
  - https://developer.android.com/reference/android/Manifest.permission

Permission: Declaration (1/3)


1 Declare the permission in your app's manifest file
2 Design your app UX to be self-explanatory
3 Wait for the user to request a specific action
4 Permission already granted to your app?
Yes
No
5a Show a rationale to the user?
Yes
No
5b Explain to the user why your app needs this permission
6 Request the permission to show the system dialog
7 Does the user grant permission to your app?
Yes
No
8a Access the info that's protected by the permission
8b Gracefully degrade your app's experience

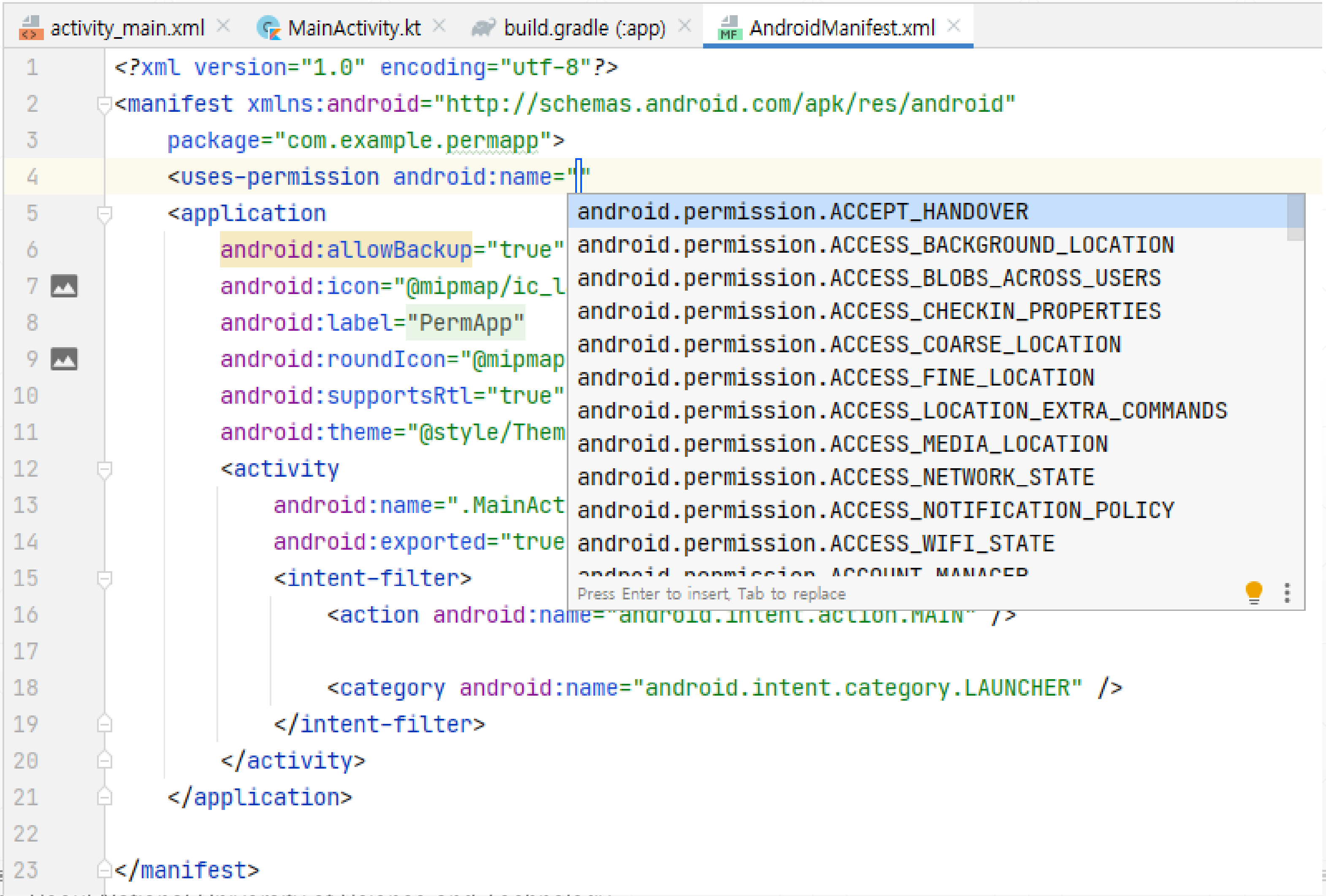
- If your app requests app permissions, you must declare these permissions in your app's manifest file
  - These declarations help app stores and users understand the set of permissions that your app might request

- Install-time permission (a normal permission or a signature permission)
  - The permission is granted automatically at install time

- Runtime permission
  - If your app is installed on a device that runs Android 6.0 (API level 23) or higher, you must request the permission by yourself

Permission: Declaration (2/3)

- Include the appropriate <uses-permission> element in your app's manifest file



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.permapp">
    <uses-permission android:name=""
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_l
        android:label="PermApp"
        android:roundIcon="@mipmap
        android:supportsRtl="true"
        android:theme="@style/Them
        <activity
            android:name=".MainAct
            android:exported="true
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

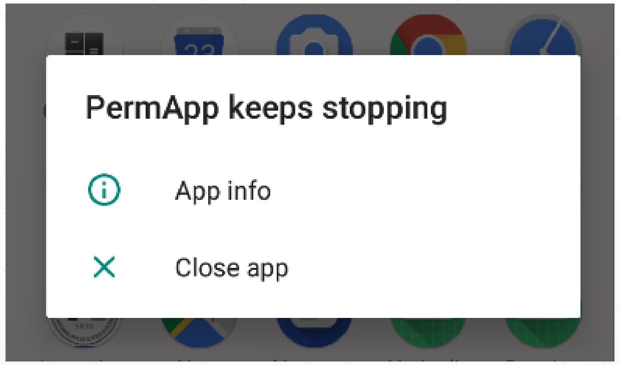
```
android.permission.ACCEPT_HANDOVER
android.permission.ACCESS_BACKGROUND_LOCATION
android.permission.ACCESS_BLOBS_ACROSS_USERS
android.permission.ACCESS_CHECKIN_PROPERTIES
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
android.permission.ACCESS_MEDIA_LOCATION
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_NOTIFICATION_POLICY
android.permission.ACCESS_WIFI_STATE
android.permission.ACCOUNT_MANAGER
Press Enter to insert, Tab to replace
```

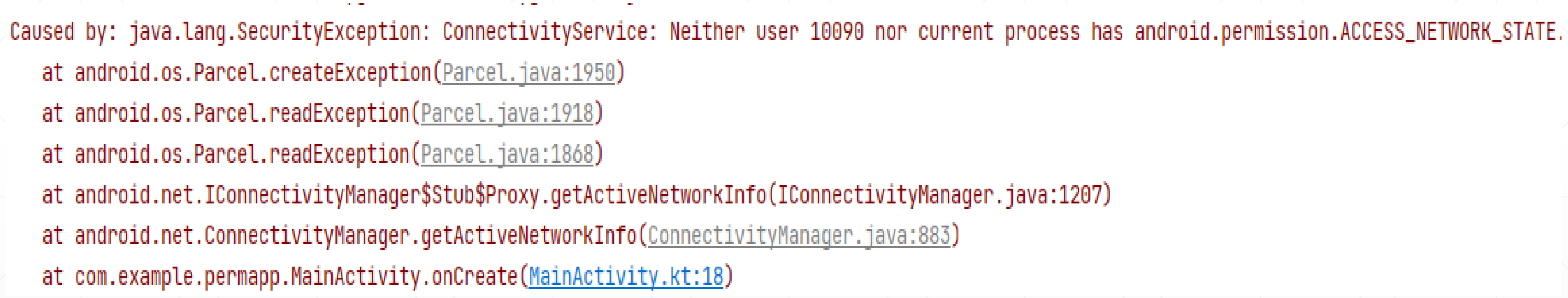
Permission: Declaration (3/3)

- Example) getting network status without proper permissions

```
val connMgr = getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
val networkInfo: NetworkInfo? = connMgr.activeNetworkInfo

Log.d("ITM", "Connected: ${networkInfo?.isConnected}")
```





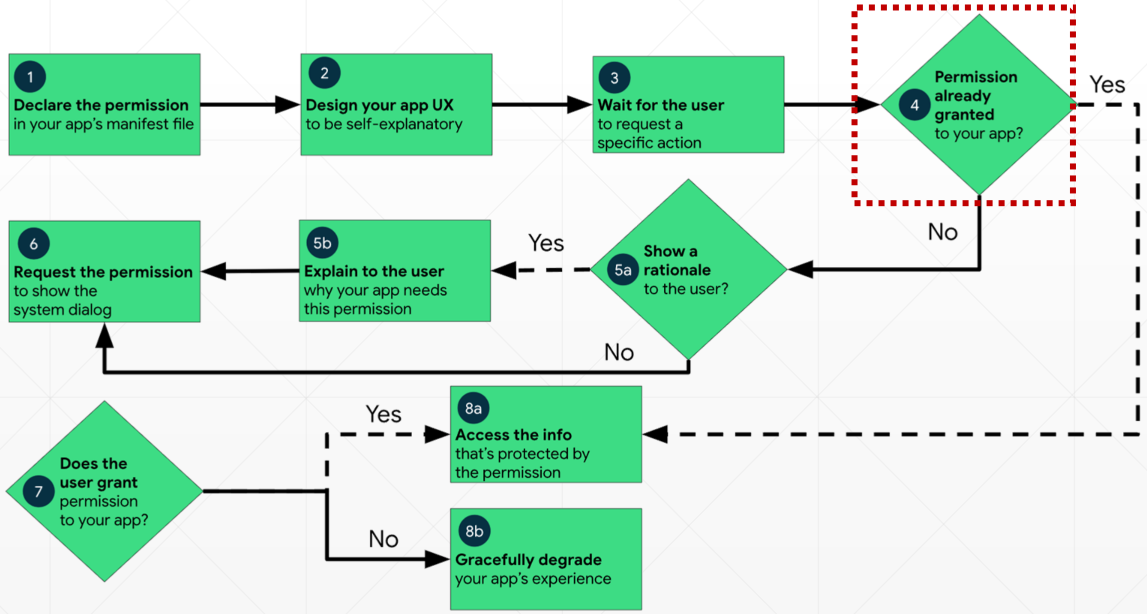
```
Caused by: java.lang.SecurityException: ConnectivityService: Neither user 10090 nor current process has android.permission.ACCESS_NETWORK_STATE.
    at android.os.Parcel.createException(Parcel.java:1950)
    at android.os.Parcel.readException(Parcel.java:1918)
    at android.os.Parcel.readException(Parcel.java:1868)
    at android.net.IConnectivityManager$Stub$Proxy.getActiveNetworkInfo(IConnectivityManager.java:1207)
    at android.net.ConnectivityManager.getActiveNetworkInfo(ConnectivityManager.java:883)
    at com.example.permapp.MainActivity.onCreate(MainActivity.kt:18)
```

➢ Add ACCESS_NETWORK_STATE permission and try again!

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

Permission: Granted?

- Determine whether your app was already granted the permission
  - Check whether the user has already granted the runtime permission that your app requires!
    - If so, your app can access the private user data
    - If not? your app needs to ask the permission
  - You must check whether you have that permission every time you perform an operation that requires that permission!



1 Declare the permission in your app's manifest file
2 Design your app UX to be self-explanatory
3 Wait for the user to request a specific action
4 Permission already granted to your app?
Yes
No
5a Show a rationale to the user?
Yes
No
5b Explain to the user why your app needs this permission
6 Request the permission to show the system dialog
7 Does the user grant permission to your app?
Yes
No
8a Access the info that's protected by the permission
8b Gracefully degrade your app's experience

Permission: Granted?

- Determine whether your app was already granted the permission
  - Check whether the user has already granted the runtime permission that your app requires!

- ContextCompat.checkSelfPermission(context, permission)
  - Permission: the name of the permission being checked
  - Returns: PERMISSION_GRANTED or PERMISSION_DENIED

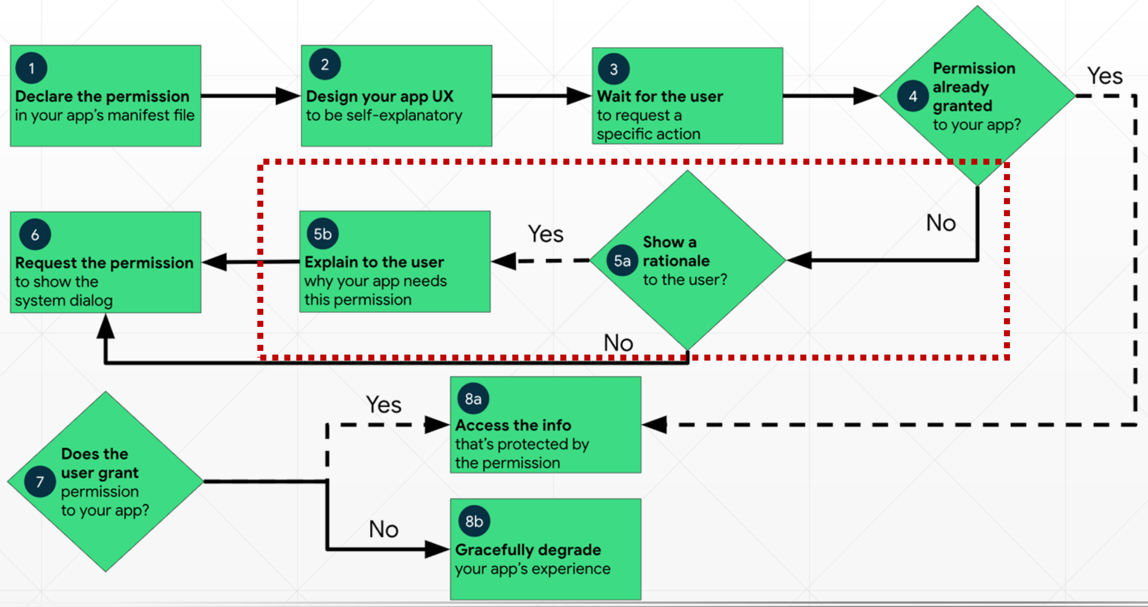
```
val perm = ContextCompat.checkSelfPermission(this, "android.permission.ACCESS_NETWORK_STATE")
if (perm == PERMISSION_GRANTED) {
    Log.d("ITM", "Permission granted!")
    val connMgr = getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    val networkInfo: NetworkInfo? = connMgr.activeNetworkInfo
    Log.d("ITM", "Connected: ${networkInfo?.isConnected}")
}
```

Permission: Request (1/7)

- Explain why your app needs the permission
  - If the ContextCompat.checkSelfPermission() method returns PERMISSION_DENIED
    - Call shouldShowRequestPermissionRationale()!
  - If shouldShowRequestPermissionRationale() returns true, show an educational UI to the user!
    - This happens when the user already denied the permission request!
    - So, describe why the feature, which the user wants to enable, needs a particular permission

…when people know why an app is using something as sensitive as their location — for example, for targeted advertising — it makes them more comfortable than when simply told an app is using their location.

Professor Jason Hong from CMU


1 Declare the permission in your app's manifest file
2 Design your app UX to be self-explanatory
3 Wait for the user to request a specific action
4 Permission already granted to your app?
Yes
No
5a Show a rationale to the user?
Yes
No
5b Explain to the user why your app needs this permission
6 Request the permission to show the system dialog
7 Does the user grant permission to your app?
Yes
No
8a Access the info that's protected by the permission
8b Gracefully degrade your app's experience

Permission: Request (2/7)

- Explain why your app needs the permission
  - Add the following runtime permissions in your manifest file

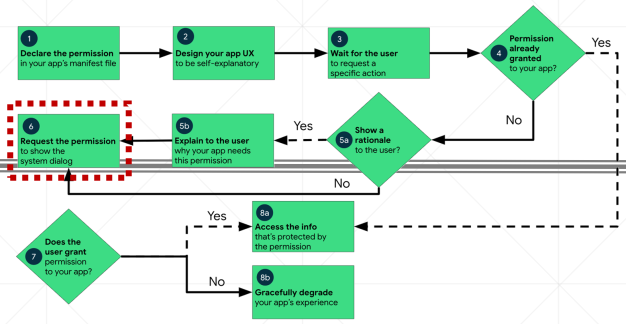
```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

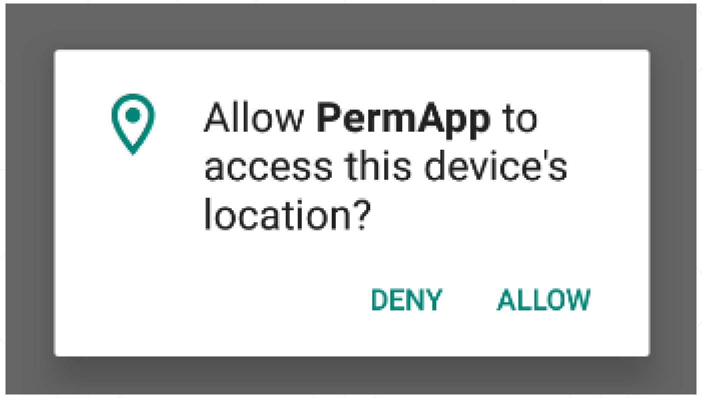
  - Code example)

```
val perm = ContextCompat.checkSelfPermission(this, "android.permission.ACCESS_FINE_LOCATION")
if (perm == PERMISSION_GRANTED) {
    val locationManager = getSystemService(LOCATION_SERVICE) as LocationManager
    Log.d("ITM","${locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER)}")
}
else{
    val permRationale = shouldShowRequestPermissionRationale("android.permission.ACCESS_FINE_LOCATION")
    Log.d("ITM","$permRationale")
}
```

  - Works fine?

Permission: Request (3/7)

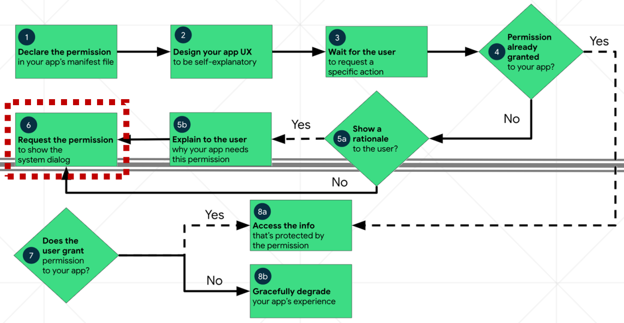

1 Declare the permission in your app's manifest file
2 Design your app UX to be self-explanatory
3 Wait for the user to request a specific action
4 Permission already granted to your app?
Yes
No
5a Show a rationale to the user?
Yes
No
5b Explain to the user why your app needs this permission
6 Request the permission to show the system dialog
7 Does the user grant permission to your app?
Yes
No
8a Access the info that's protected by the permission
8b Gracefully degrade your app's experience


Allow **PermApp** to access this device's location?

DENY ALLOW

- Request (dangerous) permissions!
  - Use permission-related contracts
    - Allows the system to manage a set of permission request codes
  - RequestPermission() contract
    - Takes a permission string as input
    - Returns Boolean value
  - RequestMultiplePermissions() contract
    - Takes an array of permission strings as input
    - Returns an array of Boolean values

Permission: Request (4/7)



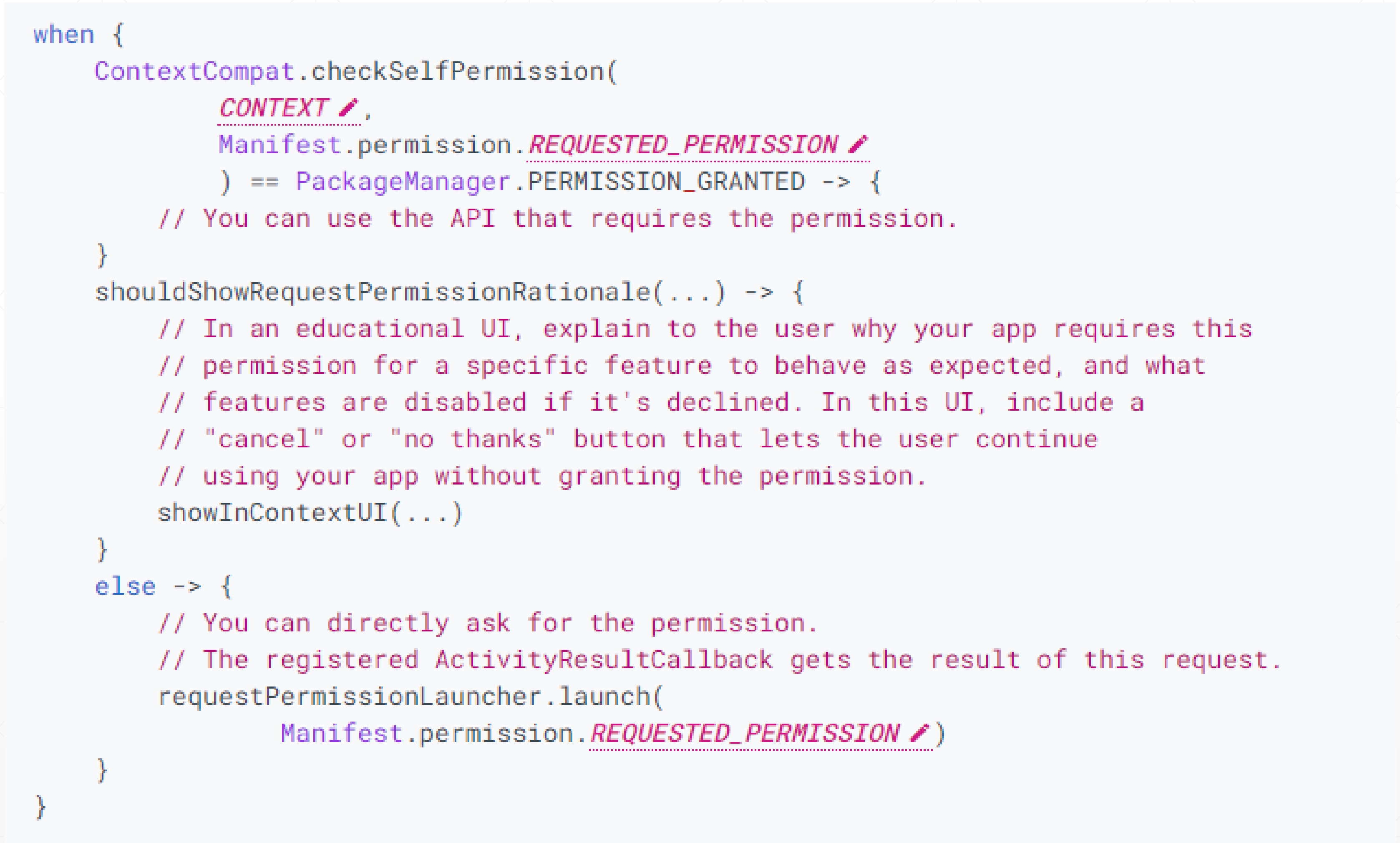
1 Declare the permission in your app's manifest file
2 Design your app UX to be self-explanatory
3 Wait for the user to request a specific action
4 Permission already granted to your app?
Yes
No
5a Show a rationale to the user?
Yes
5b Explain to the user why your app needs this permission
No
6 Request the permission to show the system dialog
7 Does the user grant permission to your app?
Yes
8a Access the info that's protected by the permission
No
8b Gracefully degrade your app's experience

- Request (dangerous) permissions!
  - Launcher & contract

```
val requestPermissionLauncher =
    registerForActivityResult(RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            // Permission is granted. Continue the action or workflow in your
            // app.
        } else {
            // Explain to the user that the feature is unavailable because the
            // feature requires a permission that the user has denied. At the
            // same time, respect the user's decision. Don't link to system
            // settings in an effort to convince the user to change their
            // decision.
        }
    }
```

Permission: Request (5/7)

- Request (dangerous) permissions!
  - Workflow



```
when {
    ContextCompat.checkSelfPermission(
            CONTEXT,
            Manifest.permission.REQUESTED_PERMISSION
            ) == PackageManager.PERMISSION_GRANTED -> {
        // You can use the API that requires the permission.
    }
    shouldShowRequestPermissionRationale(...) -> {
        // In an educational UI, explain to the user why your app requires this
        // permission for a specific feature to behave as expected, and what
        // features are disabled if it's declined. In this UI, include a
        // "cancel" or "no thanks" button that lets the user continue
        // using your app without granting the permission.
        showInContextUI(...)
    }
    else -> {
        // You can directly ask for the permission.
        // The registered ActivityResultCallback gets the result of this request.
        requestPermissionLauncher.launch(
                Manifest.permission.REQUESTED_PERMISSION)
    }
}
```

Permission: Request (6/7)

- Request (dangerous) permissions!

```
val binding by lazy { ActivityMainBinding.inflate(layoutInflater) }
lateinit var permLauncher: ActivityResultLauncher<String>

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(binding.root)

    permLauncher =
        registerForActivityResult(ActivityResultContracts.RequestPermission()) { isGranted ->
            if (isGranted) {
                Log.d(
                    "ITM",
                    "Now, permission granted by the user"
                )
            }
            else{
                Log.d("ITM","permission request denied. next time, we need to explain WHY.")
            }
        }
...
```

Permission: Request (7/7)

- Request (dangerous) permissions!

NOTE: If you run this app on the device with version >= 31,
pressing Back button in the root activity does not close the activity.
Therefore, you need to manually close the app, and re-open it to invoke onCreate() callback.

```
when {
    ContextCompat.checkSelfPermission  (
        this, "android.permission.ACCESS_FINE_LOCATION") == PERMISSION_GRANTED -> {
        val locationManager = getSystemService(LOCATION_SERVICE) as LocationManager
        Log.d(
            "ITM",
            "${locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER)}"
        )
    }
    shouldShowRequestPermissionRationale("android.permission.ACCESS_FINE_LOCATION") -> {
        Log.d("ITM", "without this? you cannot use our app :D")
    }
    else -> {
        Log.d("ITM", "request permission")
        permLauncher.launch(android.Manifest.permission.ACCESS_FINE_LOCATION)
    }

}
```

Best Practices: Google Recommendation

- Android goals related to the user privacy
  - Control: the user has control over the data that they share with apps
  - Transparency: the user understands what data an app uses, and why the app accesses this data
  - Data minimization: an app accesses and uses only the data that's required for a specific task or action that the user invokes

- Practice 1: Request a minimal number of permissions
  - Your app should request only the permissions that it needs to complete that action

- Practice 2: Associate runtime permissions with specific actions
  - Request permissions as late into the flow of your app's use cases as possible

Best Practices: Google Recommendation

- Practice 3: Consider your app's dependencies
  - When you include a library, you also inherit its permission requirements
- Practice 4: Be transparent
  - When you make a permissions request, be clear about what you're accessing, and why, so users can make informed decisions
- Practice 5: Make system accesses explicit
  - When you access sensitive data or hardware, such as the camera or microphone, provide a continuous indication in your app if the system does not provide indicators