# Mobile Applications
# CSCI 448
# Lecture 26

## Permissions & CameraX

## Learning Outcomes For Today

- Explain why an app needs to request permission for various features
- List features that require permission to use
- Discuss the difference between Permission Levels and how each are handled

# On Tap For Today

- Permissions & Android Versions
- Permission Levels
- Requesting Permission

- CameraX

- Practice

# On Tap For Today

- Permissions & Android Versions
- Permission Levels
- Requesting Permission
- CameraX
- Practice

# Security In Android

- Each app runs with a distinct system identity (Linux user ID and group ID); has its own process “sandbox”
- No app has default permission to perform any operation that would affect other apps, the OS, or the user, such as
  - Accessing user’s private data (contacts, emails)
  - Reading and writing another app’s files
- *Apps must explicitly ask for permission to share resources*

## Permission Examples

- `INTERNET`
  - If your application wishes to access the Internet through any means from your own process, e.g. using raw Java sockets

- `ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION`
  - To determine where the device is

## How to Request a Permission

- Put a `<uses-permission>` element in the `AndroidManifest.xml` file

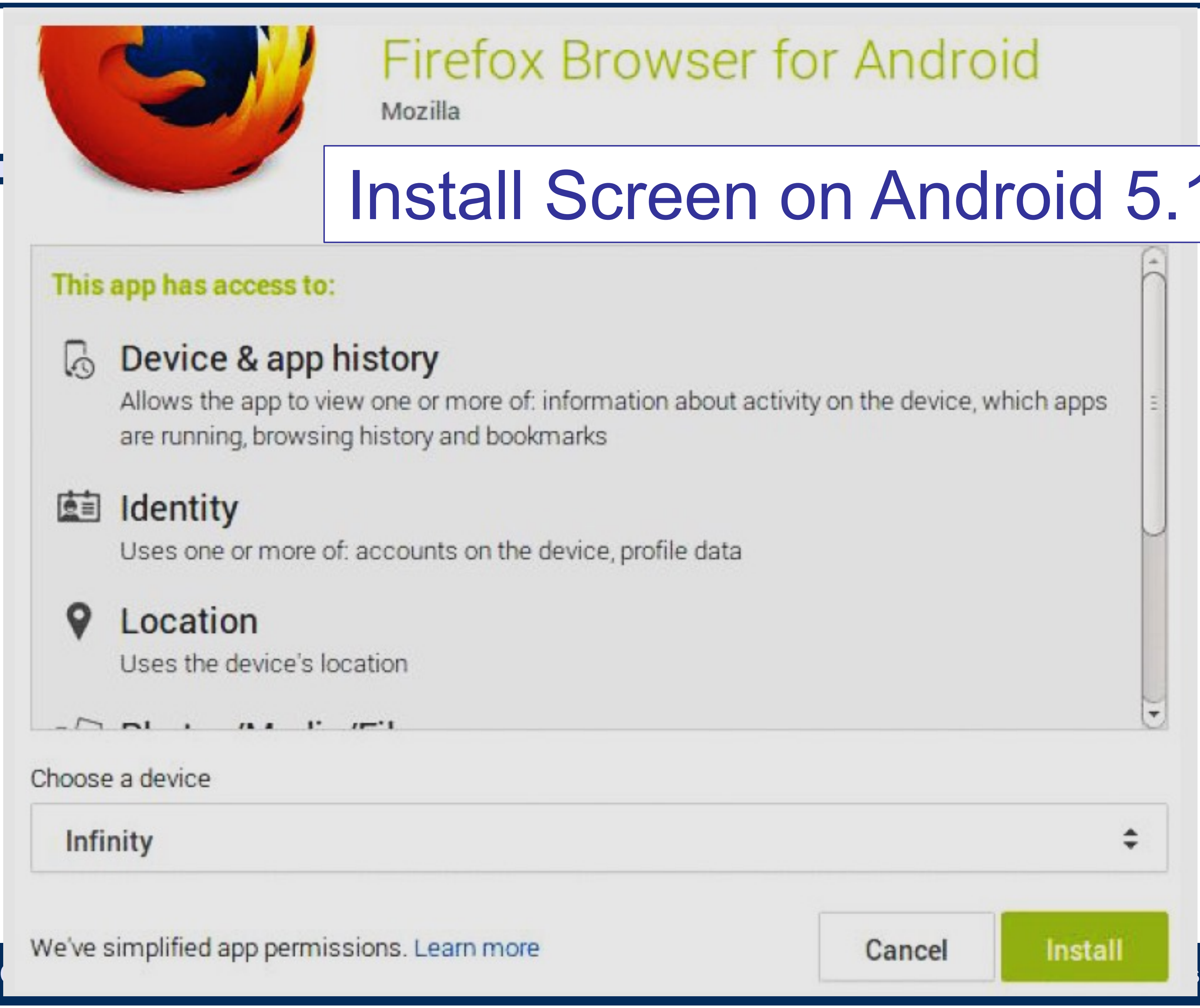
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
  package="com.commonsware.android.permmonger"
  xmlns:android="http://schemas.android.com/apk/res/android">

  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
  <uses-permission
            android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

  <application
    android:allowBackup="false"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name">
    <activity
      android:name=".MainActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>

</manifest>
```

*From: The Busy Coder's Guide to Android Development, by Mark Murphy*

# Asking User for Permissions

- When installing app through SDK tools (e.g., Android Studio) for Android 5.1 and older
  - User is not prompted for permissions
- When installing app from Play Store, for Android 5.1 and older
  - User is presented with a list of permissions; must accept all to allow install to proceed

# Install Screen on Android 5.1



Firefox Browser for Android

Mozilla

This app has access to:

**Device & app history**

Allows the app to view one or more of: information about activity on the device, which apps are running, browsing history and bookmarks

**Identity**

Uses one or more of: accounts on the device, profile data

**Location**

Uses the device's location

Choose a device

Infinity

We've simplified app permissions. Learn more

# Asking User for Permissions

- When installing app from SDK Tools or Play Store, for Android 6.0+
  - If `targetSdkVersion` is 22 or lower, you get the same behavior as for Android 5.1 and older
  - If `targetSdkVersion` is 23 or higher, the user is not prompted about permissions at install time. Instead these prompts will occur when the user runs your app and when your app asks the user for the permissions

## Why Runtime Permissions?

- Developers might need some controversial permission (e.g., READ_CONTACTS) for some fringe feature
- Prior to 6.0, they would need to request the permission from everyone, not just those who use the feature
- The runtime permission system lets them not bother the user until they need the secured feature

# More Recent Changes

- At runtime, options are:
  - Android 10
    - Only while using the app
    - Use in the background
  - Android 11
    - Just this time
    - While using the app

## On Tap For Today

- Permissions & Android Versions
- Permission Levels
- Requesting Permission

- CameraX

- Practice

## Protection Levels: Normal

- No great risk to privacy or security - user probably won't care.
- Still need to request the permission in the manifest, but system automatically grants (user not prompted).

**ACCESS_NETWORK_STATE**

**ACCESS_WIFI_STATE**

**BLUETOOTH**

**CHANGE_NETWORK_STATE**

**CHANGE_WIFI_STATE**

**DISABLE_KEYGUARD**

**EXPAND_STATUS_BAR**

**GET_PACKAGE_SIZE**

**INSTALL_SHORTCUT**

**INTERNET**

**:**

## Protection Levels: Dangerous

- Data or resources that involve the user's private information, or could potentially affect the operation of other apps

- These are divided into groups – the system just tells the user what groups the app needs, not the individual permissions

# Dangerous 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 |

*From: The Busy Coder's Guide to Android Development, by Mark Murphy*

# On Tap For Today

- Permissions & Android Versions
- Permission Levels
- Requesting Permission
- CameraX
- Practice

# Runtime Permission Checking

- `ContextCompat`
  `    .checkSelfPermission()`
  - Pass in the permission; returns true if we have

- `ActivityCompat`
  `    .shouldShowRequestPermissionRationale()`
  - Pass in the permission; returns true if user has already declined access

*Only for Android 6.0+*

# Runtime Check Flow I

```
// this is half pseudo-code and half real-code.
// pseudo is in italics
if( ContextCompat.checkSelfPermission( context, PERMISSION )
          == PackageManager.PERMISSION_GRANTED ) {
    // have permission, do task, start activity, etc.
} else {
    // don't have permission, request it...maybe







}
```

# Runtime Check Flow II

```
// this is half pseudo-code and half real-code.
// pseudo is in italics
if( ContextCompat.checkSelfPermission( context, PERMISSION )
          == PackageManager.PERMISSION_GRANTED ) {
    // have permission, do task, start activity, etc.
} else {
    // don't have permission, request it...maybe
    if( ActivityCompat.shouldShowRequestPermissionRationale( activity, PERMISSION ) ) {
        // user has already said no to request,
        // provide nice rationale why they should give permission – perhaps a Toast
    } else {
        // user has not said no yet, launch permission requester


    }
}
```

# Runtime Check Flow III

```
// this is half pseudo-code and half real-code.
// pseudo is in italics
if( ContextCompat.checkSelfPermission( context, PERMISSION )
          == PackageManager.PERMISSION_GRANTED ) {
    // have permission, do task, start activity, etc.
} else {
    // don't have permission, request it...maybe
    if( ActivityCompat.shouldShowRequestPermissionRationale( activity, PERMISSION ) ) {
        // user has already said no to request,
        // provide nice rationale why they should give permission – perhaps a Toast
    } else {
        // user has not said no yet, launch permission requester
        permissionLauncher.launch( PERMISSION )
    }
}
```

# ActivityResult Use for Permissions

```
val callback = object : ActivityResultCallback< Boolean > {
    override fun onActivityResult( isGranted: Boolean ) { ... }


val permissionLauncher: ActivityResultLauncher< String > =
        registerForActivityResult(ActivityResultContracts.RequestPermission(), callback)


...
```

```
val callbackMultiple = object : ActivityResultCallback< Boolean[] > {
    override fun onActivityResult( isGrantedArray: Boolean[] ) { ... }


val multiplePermissionLauncher: ActivityResultLauncher< String[] > =
        registerForActivityResult(ActivityResultContracts.RequestMultiplePermission(),
                                  callbackMultiple)


...
```

## Guidelines

- Explain to user *why* your app wants permission
- Don’t ask the user again after a configuration change
- If user denies, handle it gracefully (e.g., just disable some features)
- If denial means that app is useless, display a kind message stating so

# Android Design Patterns

- Behavioral Patterns
  1. Command – UI Event Handling, Retrofit Request Callback, Activity Result Callback, Permissions Callback
  2. Observer – State, Flow, LiveData
  3. Template Method - IScreenSpec
- Creational Patterns
  4. Builder – Compose NavGraph, WorkRequest, Constraints, Retrofit
  5. Factory – ViewModelFactory
  6. Singleton – ViewModelProvider, Repository, Room Database
- Structural Patterns
  7. Decorator – View Model
  8. Façade – DAO, Repository

# On Tap For Today

- Permissions & Android Versions
- Permission Levels
- Requesting Permission

- CameraX

- Practice

# CameraX

- Accesses the camera(s) on the device
- `android.camera` API
  - Can preview images
  - Can capture images
  - Can analyze images as they come in from camera
  - Can capture video (with or without audio)
- See https://developer.android.com/training/camerax

# CameraX Dependencies

- Current version 1.2.1

```
dependencies {
  def camerax_version = "1.2.1"
  implementation "androidx.camera:camera-core:${camerax_version}"
  implementation "androidx.camera:camera-camera2:${camerax_version}"
  implementation "androidx.camera:camera-lifecycle:${camerax_version}"
  implementation "androidx.camera:camera-video:${camerax_version}"

  implementation "androidx.camera:camera-view:${camerax_version}"
  implementation "androidx.camera:camera-extensions:${camerax_version}"
}
```

## CameraX Permissions

- `Manifest.permission.CAMERA`
- `Manifest.permission.RECORD_AUDIO` (optional)

## CameraX

- Get the camera provider
- Select which camera (front/back)
- Bind camera to activity lifecycle

- Then attach applicable use cases

# Preview

- Use viewfinder to preview image before capturing

- See https://developer.android.com/codelabs/camerax-getting-started#3 & https://www.kiloloco.com/articles/015-camera-jetpack-compose/

# Image Capture

- Performs `takePicture()` action and does one of
  - Returns image in memory
  - Saves image to disk
    - Specify location to write to

- See https://developer.android.com/codelabs/camerax-getting-started#4

## Other Use Cases

- Image Analysis https://developer.android.com/codelabs/camerax-getting-started#5
- Video Capture https://developer.android.com/codelabs/camerax-getting-started#6

## !!! Important Note !!!

- When using CameraX with the emulator
  - Edit emulator
  - Advanced Settings
  - Cameras must be set to **`Emulated`** instead of **`Virtual Scene`**

- Bug within emulator ☹

## Tutorials

- Android CodeLab
  - https://developer.android.com/codelabs/camerax-getting-started

- Using Jetpack Compose
  - https://www.kiloloco.com/articles/015-camera-jetpack-compose/

## On Tap For Today

- Permissions & Android Versions
- Permission Levels
- Requesting Permission
- CameraX
- Practice

# To Do For Next Time

- Alpha Feedback due Fri Mar 17
- Lab08 due Sat Mar 18
- !!! Spring Break !!!
- Lab09 due Tue Mar 28

- When we get back:
  - Location, Map
  - Notifications
  - UX/UI
  - Firebase: Remote Database, User Authentication
  - Deploying