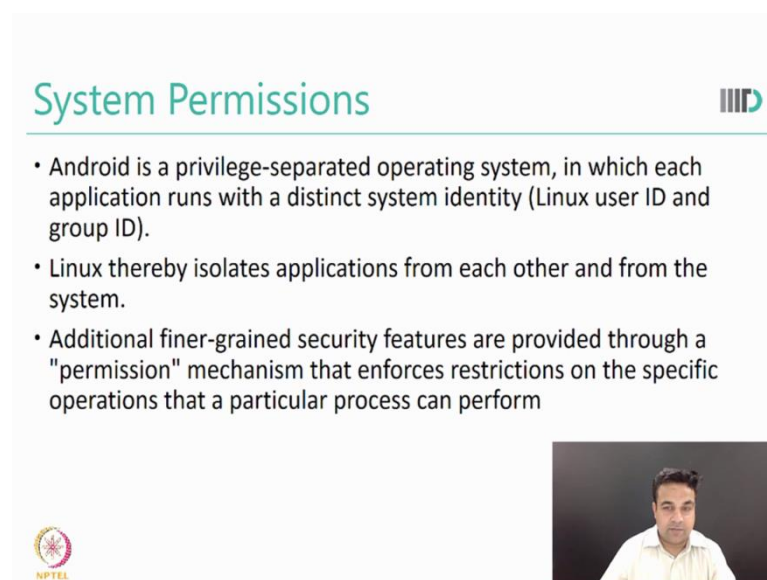


Mobile Computing
Professor Pushendra Singh
Indraprastha Institute of Information Technology Delhi
Lecture 30
System Permissions

(Refer Slide Time: 0:25)



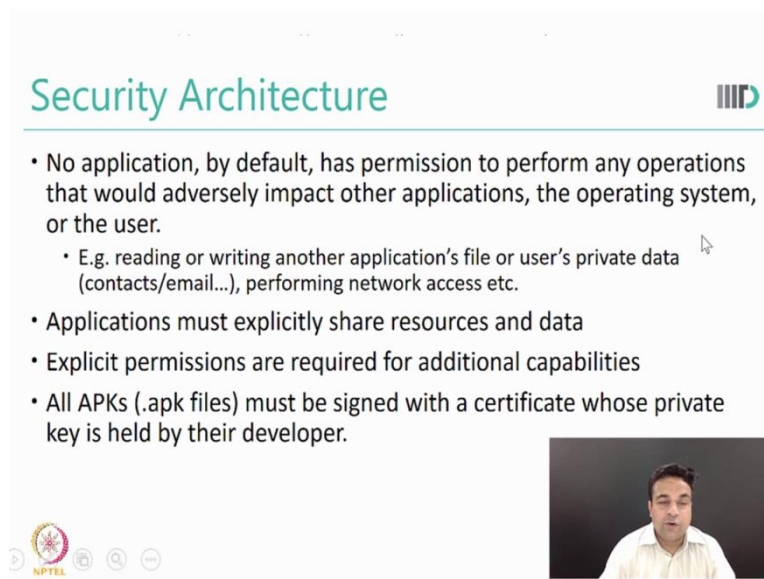
The slide is titled "System Permissions" in a teal font. It features a list of three bullet points explaining Android's privilege-separated architecture. In the bottom right corner, there is a small video inset showing the professor speaking. The NPTEL logo is visible in the bottom left corner of the slide.

- Android is a privilege-separated operating system, in which each application runs with a distinct system identity (Linux user ID and group ID).
- Linux thereby isolates applications from each other and from the system.
- Additional finer-grained security features are provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform

Hello, today we will learn about System permissions. As we have discussed already that android is a privilege-separated operating system that is that in the android each applications runs with a distinct system identity which are the Linux user ID and group ID. In terms of traditional operating system each application is a different user of the android operating system. And just like in the tradition Linux operating system each user is separated from another user. In android operating system each application is separated from another application. So this is the way the Linux which is under the android isolates application from each other and from the system as well. Besides this isolation the additional finer-grained security features are provided through a permission mechanism.

The permission mechanism enforces restrictions on the specific operations that a particular process can perform. You are already slightly familiar with the permission mechanism, because if you see in the Linux for example every file has some permission (()) (1:43). And these permission show that who can do what with a file. So some may have permissions to only read, some may have permissions to read and write, some may have only permissions to write, some may have permission to execute as well, so android operating system is also something similar for applications.

(Refer Slide Time: 2:13)



The slide is titled "Security Architecture" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by a stylized 'D'. The main content is a list of four bullet points:

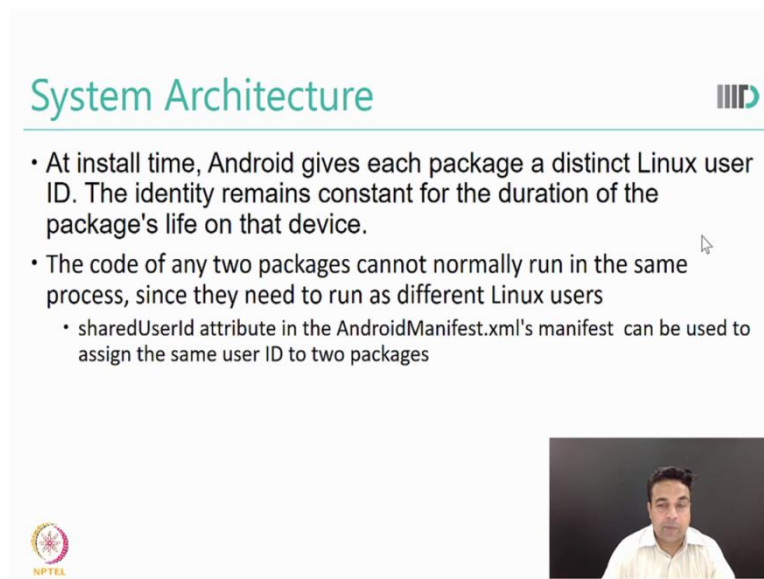
- No application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user.
 - E.g. reading or writing another application's file or user's private data (contacts/email...), performing network access etc.
- Applications must explicitly share resources and data
- Explicit permissions are required for additional capabilities
- All APKs (.apk files) must be signed with a certificate whose private key is held by their developer.

In the bottom right corner, there is a small video inset showing a man in a light-colored shirt speaking. In the bottom left corner, there are several small icons, including one with the letters "NPTEL".

Now, in android no application by default has permission to perform any operation that would adversely impact other applications, or the operating system, or the user. For example, let us say reading, writing another applications file or reading or writing users private data such as contacts. Now user's contacts without any doubt are a private data for the user, but more than that another application may also be using user's contacts. So when an application wants to perform any operation on the contacts it must ask for a permission that looks very duty.



Similarly applications must explicitly share resources and data otherwise everything is private to an application. An explicit permission will always be required for additional capabilities. There would never be a case when the additional capabilities are given without explicit permissions. Also all APK files in android must be signed with the certificate and the developer must hold the private key of that certificate, so this is the basic security architecture of the android operating system.

(Refer Slide Time: 3:44)



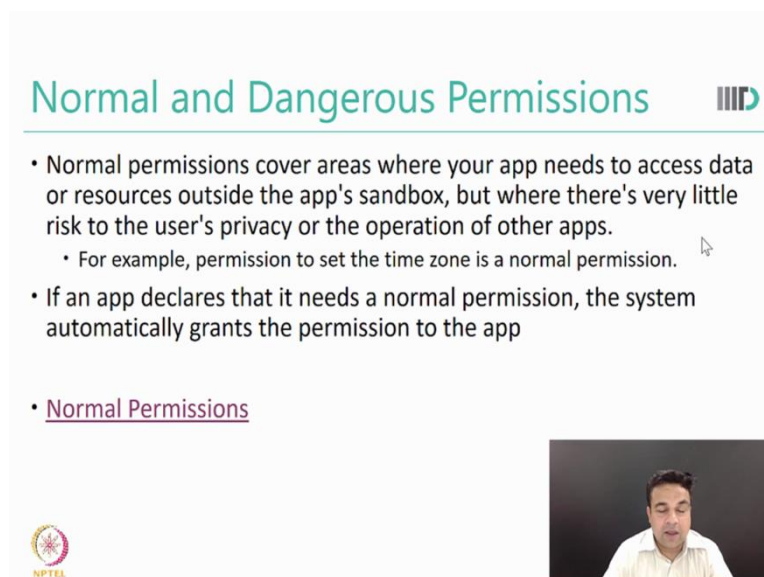
System Architecture

- At install time, Android gives each package a distinct Linux user ID. The identity remains constant for the duration of the package's life on that device.
- The code of any two packages cannot normally run in the same process, since they need to run as different Linux users
 - sharedUserId attribute in the AndroidManifest.xml's manifest can be used to assign the same user ID to two packages





So at install time android will give each package and remember package that we give in the beginning of lecture. So android will give each package a distinct Linux user id and this identity will remain constant for the duration of the package life on that device. Now any two packages cannot normally run in the same process because they need to be isolated. However if you do want this functionality then you can set an attribute called shared user ID in the Android Manifest file and then you can have 2 packages using the same user ID. Later in the course you will see when this condition is suitable and how to use it.

(Refer Slide Time: 4:38)



Normal and Dangerous Permissions

- Normal permissions cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps.
 - For example, permission to set the time zone is a normal permission.
- If an app declares that it needs a normal permission, the system automatically grants the permission to the app
- [Normal Permissions](#)

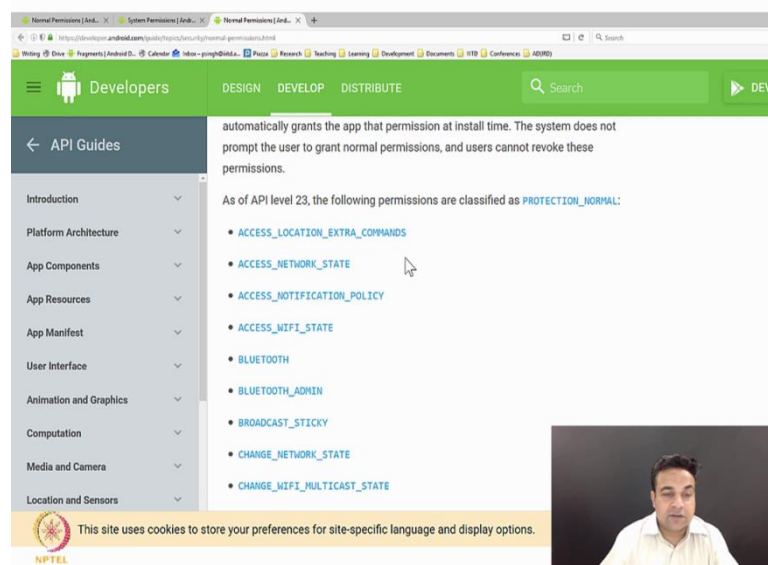


Now coming with to permissions there are two types of permissions one are normal permissions and the other are dangerous permissions. Now normal permissions are the

permissions that our areas where your app needs to access data resource outside the apps sandbox that was which is definitely accessing something outside the sandbox, but whatever it is accessing possess no or very little risk to users privacy or the operation of other apps, so one simple example is setting the time zone. Now when it is setting a time zone, when an app is setting a time zone it is definitely re-accessing data or resources which are outside the app sandbox however, as we know that setting the time zone may not have a much of impact on users privacy or the operations for any other application and that is why we are calling such permissions as normal permissions.

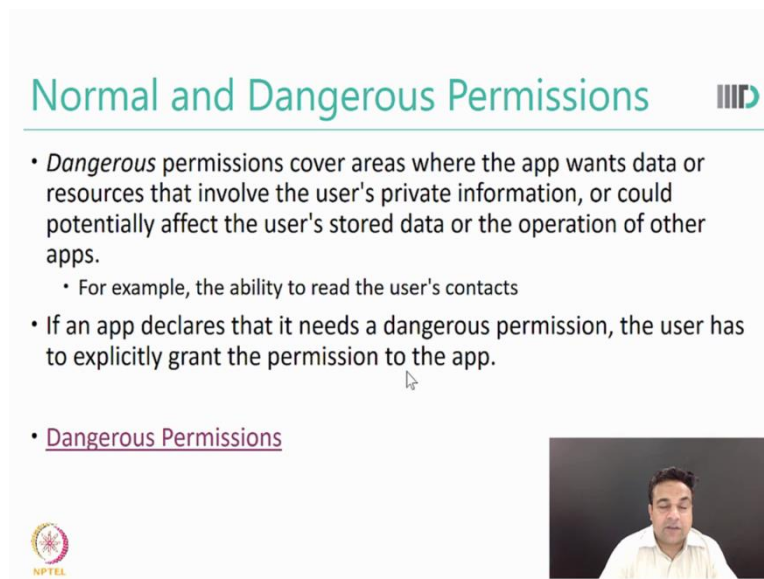
Now if an app declares that it needs a normal permission the system automatically grants the permission to the app because system is sure that granting this permission will neither impact users privacy and nor operation of other apps. So let us see the example of some of the normal permissions, which are defined in the android system.

(Refer Slide Time: 6:16)



So here is a list of normal permissions given on developer.android.com so for example ACCESS_LOCATION_EXTRA_COMMANDS, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE that is where the Wi-Fi is on not, INTERNET, REORDER_TASKS, SETTING_ALARM, SETTING_TIME_ZONE, etc, etc and as you can go through the list you can find out that yes indeed these permission do not seem to reposing or seem to reposing a very little privacy list, that is why they are branded under the normal permissions.

(Refer Slide Time: 6:56)



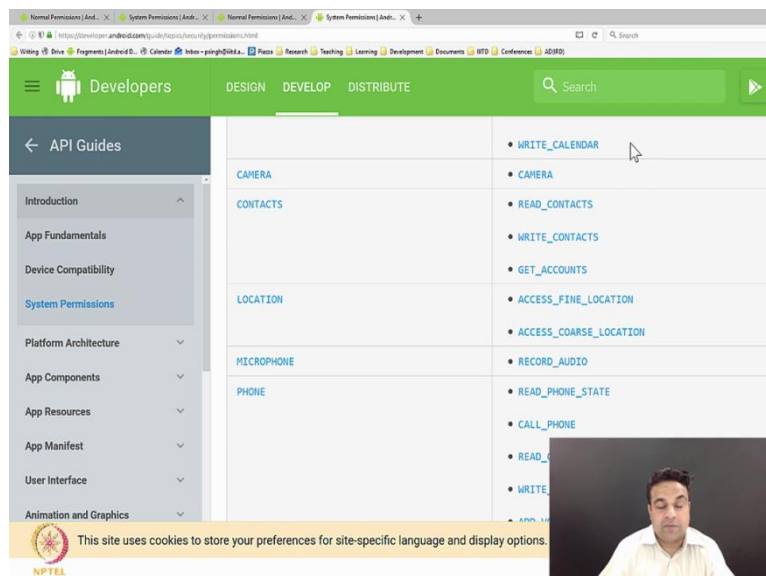
The slide is titled "Normal and Dangerous Permissions" in a teal font. It features a list of bullet points explaining dangerous permissions. A mouse cursor is visible over the text. In the bottom right corner, there is a small video inset showing a man speaking. In the bottom left corner, there is a small circular logo with the text "NPTEL" below it.

- *Dangerous* permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps.
 - For example, the ability to read the user's contacts
- If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app.
- Dangerous Permissions

The second type of permissions are dangerous permissions and dangerous permissions as the name suggest cover areas where the app wants data or resources that involve users private information or could potentially affect users stored data or the operation of other apps. Let us take an example, suppose the permission is to read the users contacts. A user's contacts are used as private information and as I said earlier another application may also be using it. So not only it impacts the privacy it also impacts stability of other applications.

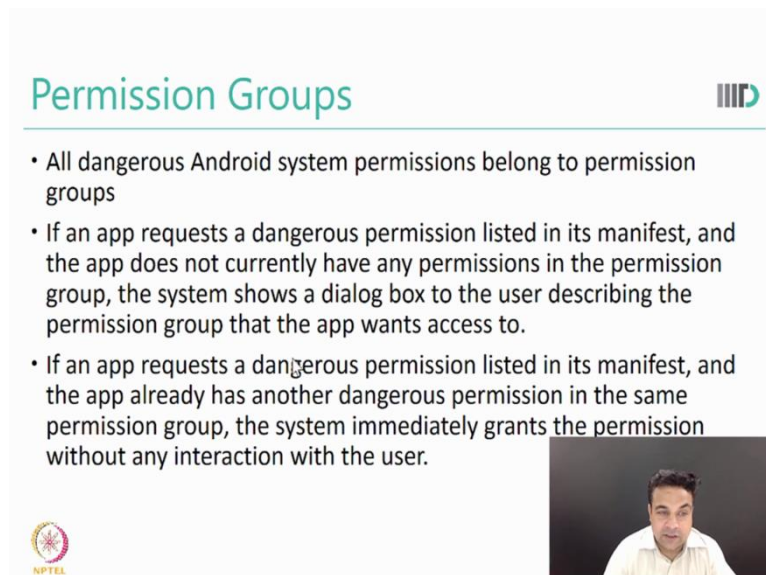
Now if an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. If you have an older android device older means older than android 5.0, then the permissions are granted at the install time. However if you have a newer android device such as android 6P or Lava android 7, then permissions are granted at the time of running the application. However, in both of the cases permissions must be explicitly granted, so let us see an example of dangerous permissions.

(Refer Slide Time: 8:31)



So here are some dangerous some permissions: WRITING_CALENDER, READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS, WRITE_CALL_LOG, READ_CALL_LOG, etc, etc, etc. When we discuss the dangerous permissions we must also discuss permission groups.

(Refer Slide Time: 8:57)

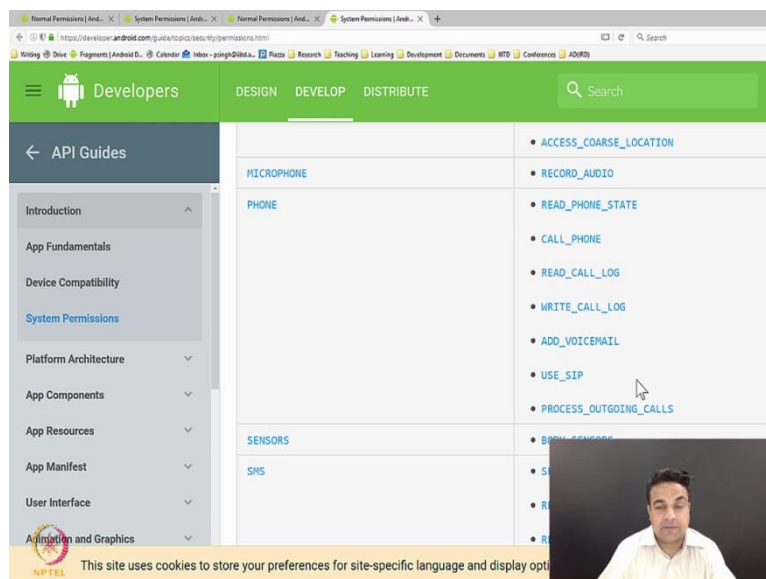


So the permission group is simply a group which contains one or more dangerous permissions. Or in other words, all dangerous android system permissions belong to permission groups that is there would not be any dangerous permission which will not belong to a permission group. Now if an app request the dangerous permission listed in its manifest

and the app is not currently have any permissions in the permission group, the system shows a dialog box to the user describing the permission group that the app wants access to.

On other hand, if an app requests a dangerous permission listed in its manifest, and the app already has another dangerous permission in the same permission group, the system immediately grants the permission without any interaction with the user, let us try to understand with the example that we had just now.

(Refer Slide Time: 10:02)



Now as you can see that the permission group PHONE has many permissions here. Now suppose in the very beginning my app requested a permission to READ_CALL_LOG and because by that time user has not granted permissions to any permissions in this particular permission group, the user will be prompted and will be asked to give the yes or no as per allowing to the permission is concern. However, once the user has allowed permission in the permission group of PHONE and then any other permission coming for the permission group of PHONE will be automatically granted. So if my app now uses WRITE_CALL_LOG it will be automatically granted because android assumes that if you are ok with the permission group, then you are ok with all the permissions of that permission group.

(Refer Slide Time: 11:05)

The slide titled "Permission Groups" displays a table with two columns: "Permission Group" and "Permissions".

Permission Group	Permissions
CALENDAR	<ul style="list-style-type: none">•READ_CALENDAR•WRITE_CALENDAR
CAMERA	<ul style="list-style-type: none">•CAMERA
CONTACTS	<ul style="list-style-type: none">•READ_CONTACTS•WRITE_CONTACTS•GET_ACCOUNTS

Below the table is a URL: <https://developer.android.com/guide/topics/security/permissions.htm>

The slide also features a video inset of a presenter in the bottom right corner and navigation icons in the bottom left corner.

So here is a slide giving some of the permission groups and the permission we have already seen, that on this website we can actually get the complete list. Now let us see how do we define and enforce permissions.

(Refer Slide Time: 11:25)

The slide titled "Permission Groups" displays a table with two columns: "Permission Group" and "Permissions".

Permission Group	Permissions
CALENDAR	<ul style="list-style-type: none">•READ_CALENDAR•WRITE_CALENDAR
CAMERA	<ul style="list-style-type: none">•CAMERA
CONTACTS	<ul style="list-style-type: none">•READ_CONTACTS•WRITE_CONTACTS•GET_ACCOUNTS

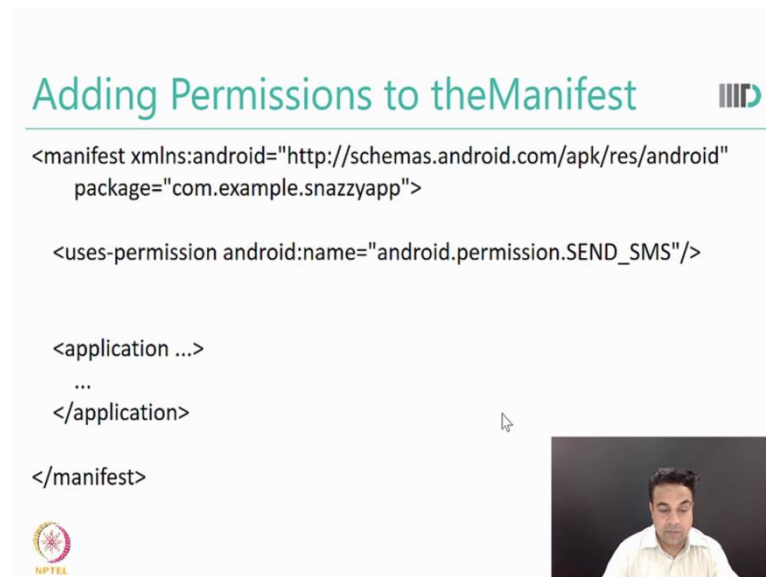
Below the table is a URL: <https://developer.android.com/guide/topics/security/permissions.htm>

The slide also features a video inset of a presenter in the bottom right corner and navigation icons in the bottom left corner.

So if you want to enforce our own permission that is that we have declared an activity and we want to make sure that no other activity can invoke it unless it ask the certain permissions. So we will have to declare this permission AndroidManifest.xml. So in the given example I have an activity called DEADLY_ACTIVITY and I want to say that this is a dangerous permission, so that is the user must explicitly grant permission to the work using this DEADLY_ACTIVITY.

In this case I write a few attributes name the most important one is the android protection Level, which I mention as dangerous. So in this case now when the when I use a request when the app request access the user will be prompted whether he wants to grant the access or not. I am also putting it inside the permission group, so if the user has already given permission whether in this permission group then he or she is not prompted otherwise they are prompted. Now let us move on to working with system preferences.

(Refer Slide Time: 12:43)



Adding Permissions to the Manifest

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

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

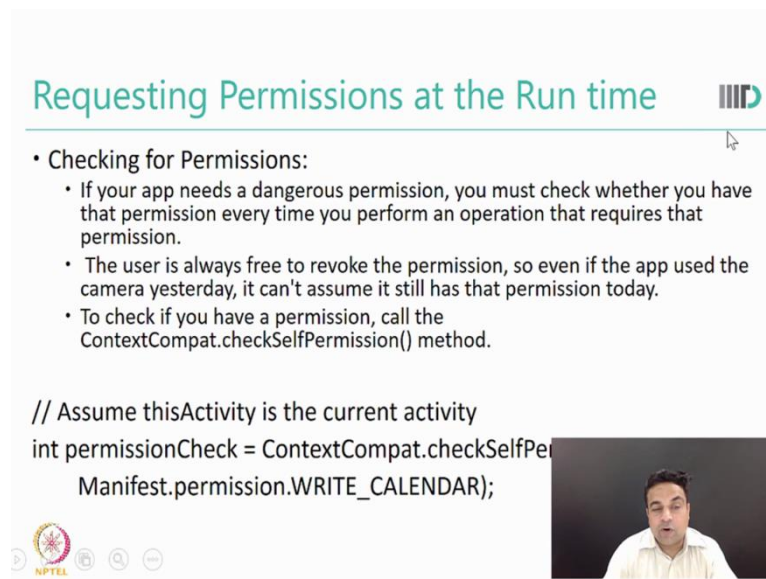
    <application ...>
        ...
    </application>

</manifest>
```

NPTEL

So the first step is to work with system preferences is to add permissions to the manifest, this is for the case when we want to have certain permissions in our app. Suppose, we want to have a permission called SEND_SMS in this case in my package of let us say this snazzyapp I would like to add a permission of SEND_SMS. Now when my app is running the user add some part of time will be prompted to allow the permission for the SEND_SMS or to read the runtime or if there is an older android device then during the install time.

(Refer Slide Time: 13:32)



The slide is titled "Requesting Permissions at the Run time" in a teal font. It features a list of bullet points under the heading "Checking for Permissions:". The code snippet shows a Java method call to check a permission. A small video inset of a man is visible in the bottom right corner of the slide.

Requesting Permissions at the Run time

- Checking for Permissions:
 - If your app needs a dangerous permission, you must check whether you have that permission every time you perform an operation that requires that permission.
 - The user is always free to revoke the permission, so even if the app used the camera yesterday, it can't assume it still has that permission today.
 - To check if you have a permission, call the `ContextCompat.checkSelfPermission()` method.

```
// Assume thisActivity is the current activity
int permissionCheck = ContextCompat.checkSelfPermission(
    thisActivity, Manifest.permission.WRITE_CALENDAR);
```

Now because the android changed its method of asking for permissions that is now the permissions are asked at the runtime instead of the install time. If your app needs a dangerous permission you must check whether you have that permission every time you perform an app operation that requires that permission. That is if you want to read users contact which is a dangerous permission before reading users contacts you must check that you still have the permission from the user. The reason for that is that the user is always free to revoke the permission, so even if your application read the contacts yesterday or use the camera yesterday you cannot assume that it is still has that permission today, that is you must make an explicit check.

Now in order to check the android makes it very easy, there is a simple method called `ContextCompat.checkSelfPermission` and using this method you can actually check whether the permission is there or not. Here is a simple code explaining it, so I run the permission check on this particular activity and with respect to permission type or permission group and I (()) (14:52) the device, so it is as simple as that.

(Refer Slide Time: 15:00)

Request Permissions

```
// Here, thisActivity is the current activity
if (ContextCompat.checkSelfPermission(thisActivity,
    Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {







    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
        Manifest.permission.READ_CONTACTS)) {

        // Show an explanation to the user *asynchronously* -- don't block
        // this thread waiting for the user's response! After the user
        // sees the explanation, try again to request the permission.
    }

} else {

    // No explanation needed, we can request the permission.
    ActivityCompat.requestPermissions(thisActivity,
        new String[]{Manifest.permission.READ_CONTACTS},
        MY_PERMISSIONS_REQUEST_READ_CONTACTS);

    // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
    // app-defined int constant. The callback method gets the
    // result of the request.
}
```









Now let us look programming code where we are trying to handle groups when the permission is granted and when the permission is not granted. So if we look `checkSelfPermission` this activity `READ_CONTACTS` and this is not equal to `PERMISSION_GRANTED` that is permission has not been granted, in such cases you may want to show the user an explanation. So for example you may want to tell the user look if you do not give the permission then may be these extra features will not be working. On the other hand, on the right side if the permission is given in then you may not want to give an explanation to the user because user has already given permission and you would like to continue with it.

(Refer Slide Time: 15:53)

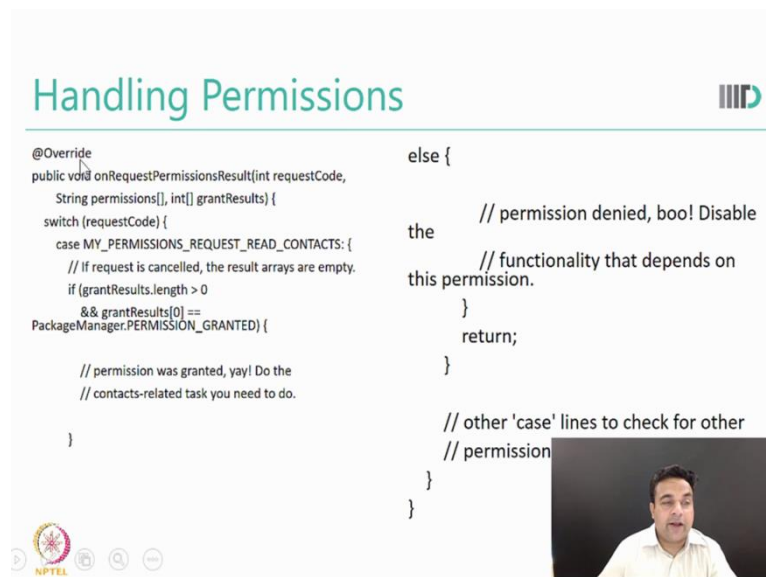
Handling the permissions

- When your app requests permissions, the system presents a dialog box to the user.
- When the user responds, the system invokes your app's `onRequestPermissionsResult()` method, passing it the user response.
- Your app has to override that method to find out whether the permission was granted.
- The callback is passed the same request code you passed to `requestPermissions()`



Now let us see how to handle the permissions. So when your app requests permissions, the system presents a dialog box to the user and the user responds. So when the user responds, the system invokes our apps `onRequestPermissionsResult()` method and it passes the user response inside this method. Now as an application developer you must override this method to find out whether the permission was granted or not and this callback is passed the same request code you passed to `requestPermissions()`. So let us quickly see a programming code to see how to get this run in a program.

(Refer Slide Time: 16:41)



Handling Permissions

```
@Override
public void onRequestPermissionsResult(int requestCode,
String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED) {

                // permission was granted, yay! Do the
                // contacts-related task you need to do.

            }

            else {
                // permission denied, boo! Disable
                the
                // functionality that depends on
                this permission.
            }
            return;
        }

        // other 'case' lines to check for other
        // permission
    }
}
```

So for example here we are overriding `onRequestPermissionsResult()` method which has three parameters `requestCode`, `String permissions` and `grantResults`. And depending on the `requestCode` that is I am putting switch statement and I am then making a case `MY_PERMISSIONS_REQUEST_READ_CONTACTS` and then we can say here you know what is the result is if the result is cancelled the result arrays are empty, do some piece of code and if the permission is divided then do some other piece of code. So essentially you should be able to write the code for the `(())` (17:19).

Now this was all about permissions, now let us discuss something very important, which is the best practices of using permissions. number 1 why let us go back and think that why do we need permissions. So we need permissions when we want to do something which impacts a users experience for example you may want to take a camera take a photo using a camera or you may want to read users contacts.

(Refer Slide Time: 18:24)

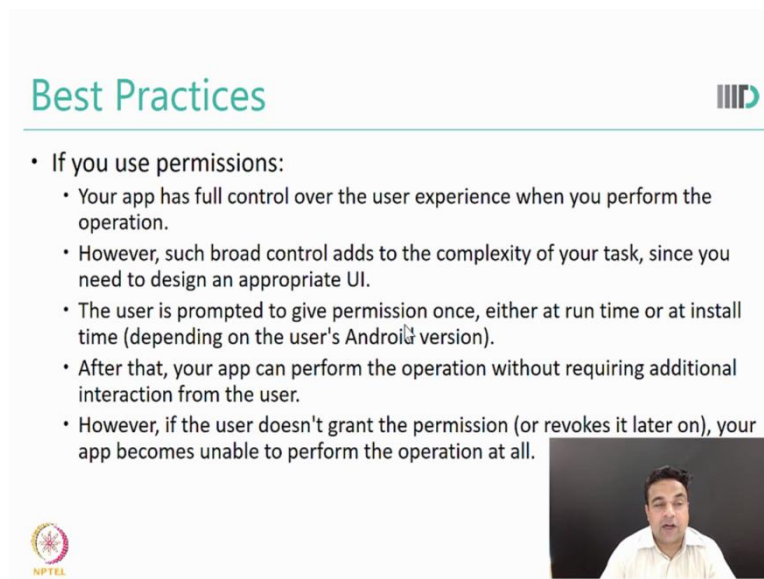
The image shows a presentation slide with the following elements:

- Title:** Best Practices
- Bullet Point:** • Consider using an Intent
- Logos:** NPTEL (National Programme on Technology Enhanced Learning) logo in the bottom left and IITD (Indian Institute of Technology Delhi) logo in the top right.
- Video Inset:** A small video window in the bottom right corner showing a man in a light-colored shirt speaking.

Now in all such cases we are actually effectively invoking another activity of another application. So when we want to execute the camera, there is a camera app which can also be used. So we have two possibilities, number one we can ask a permission to use the camera and hope that the user grants the permission so that we can use the camera or we can simply use an intent. Like in the last few lectures we studied that intents specifically implicit intents are used to know activities in other applications and that is and these applications may be system application.

So I can use intent to invoke the camera application. In this case I will not have to ask for permissions. So just to recap I had a choice either asks for permissions to use the camera or user intent for using the camera. For intent, no permission required permissions permission which is required. Now let us see what happens when we use permissions.

(Refer Slide Time: 19:15)



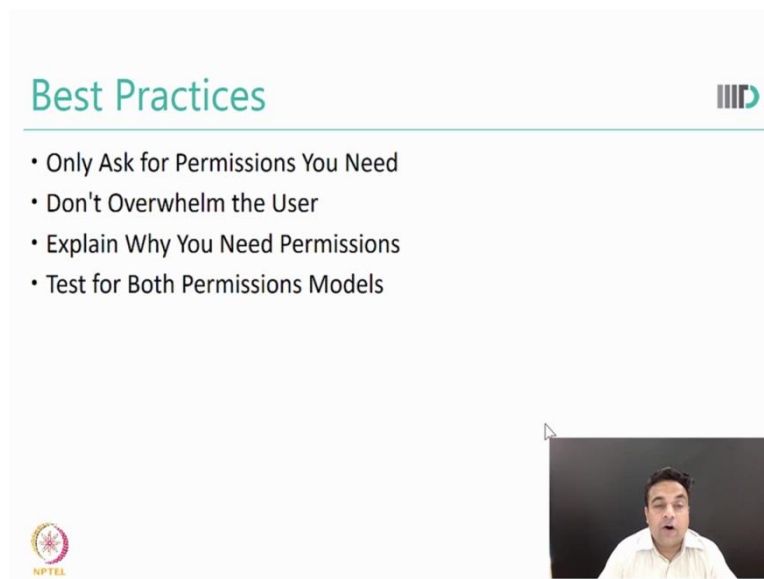
The slide is titled "Best Practices" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by the letter "D". Below the title, there is a list of bullet points. The first bullet point is "If you use permissions:", followed by five sub-bullets. The sub-bullets describe how permissions affect user experience, UI design, permission prompts, and the consequences of not granting permissions. In the bottom right corner of the slide, there is a small video inset showing a man in a light-colored shirt speaking. In the bottom left corner, there is a small circular logo with the text "NPTEL" below it.

- If you use permissions:
 - Your app has full control over the user experience when you perform the operation.
 - However, such broad control adds to the complexity of your task, since you need to design an appropriate UI.
 - The user is prompted to give permission once, either at run time or at install time (depending on the user's Android version).
 - After that, your app can perform the operation without requiring additional interaction from the user.
 - However, if the user doesn't grant the permission (or revokes it later on), your app becomes unable to perform the operation at all.

So when we use permissions our app has full control over the user experience that is we will have to design an appropriate UI so even if we are using camera app it will run within our UI if we are running it using permissions. Now this may add the complexity but it also gives you more freedom. Now if the user does not grant the permission then in that case our application becomes unable to perform the operation at all. So if there is a very critical requirement for some permission which are does not granted our application will have to stop right there this is the disadvantage of using permissions.

Now let us see what if we use intent. So if we use intent we do not have to design the UI for the operation. So in the intent if I invoke a camera app, the camera app will be invoked and camera app will have its own UI that is the app that handles the intent provides us the UI. So we do not have to bother about it. However, in some cases it may also mean that do not have control over the user experience. So if my app has a certain theme and I use intent to invoke let us say camera, I cannot maintain the same theme when the camera functions. Now in this case if the user does not have a default app, then the system prompts the user to choose an app otherwise our default app is use such as if we are using chrome browser on the android, then the chrome browser will be used otherwise if we have a Firefox and chrome then user will be given a choice.

(Refer Slide Time: 21:09)



Best Practices

- Only Ask for Permissions You Need
- Don't Overwhelm the User
- Explain Why You Need Permissions
- Test for Both Permissions Models

NPTEL

Video inset showing a speaker.

So let us go back to best permissions one more time number one only ask for permissions that you need, so do not ask for extra permissions, permissions cause privacy worries and users are very reluctant to provide permissions specially when the permission are let us say reading the contacts, ok, which is a very information. Similarly, do not overwhelm the user, so if possible may be use intent if possible may be design your application so that you do not need the permission. And if at all you do need permission, may be you would like to explain to the user why you need the permission. May be you would like to say that the permission will allow extra functionality in the application or extra features of the applications.

(Refer Slide Time: 22:15)



References

- <https://developer.android.com/training/index.html>
- Android Programming: The Big Nerd Ranch Guide (2nd Edition) by Bill Philips and Chris Stewart
 - <https://www.bignerdranch.com/we-write/android-programming/>
- Core Java Volume I--Fundamentals: 1 (Core Series) by Cay S. Horstmann
 - <http://horstmann.com/corejava.html>

NPTEL

Video inset showing a speaker.

The last but not the least you must test for both permissions models that is when we get permission and when you did not get. That is all about system permissions these are the references from which I have created the slides, thank you.