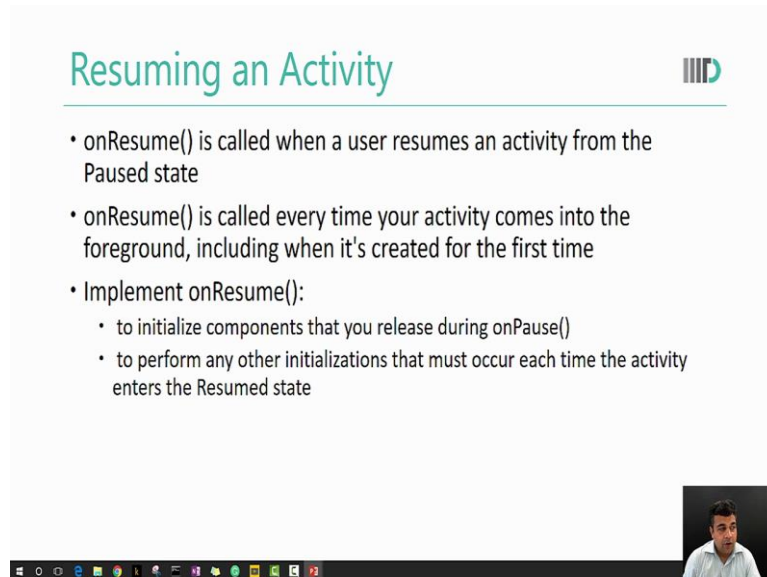


Mobile Computing
Professor Pushpedra Singh
Indraprasth Institute of Information Technology Delhi
Activity
Lecture 15

Hello in last lecture we studied different application lifecycle states. We discuss what happens when application starts and we also discuss what happens when an application paused. We will continue our lecture from that and now we will see what happens when application moves through different analytics. So let's start with resuming an activity.

(Refer Slide Time: 0:42)



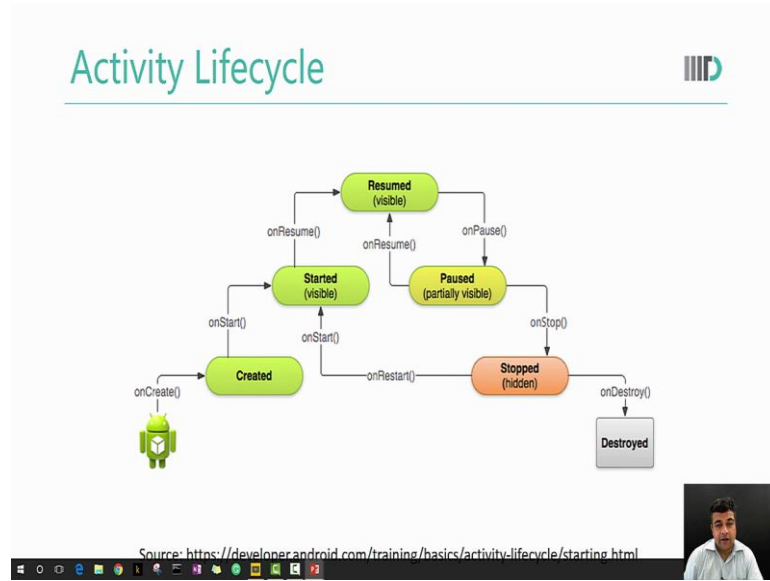
The slide is titled "Resuming an Activity" and features the IITD logo in the top right corner. It contains a list of bullet points:

- `onResume()` is called when a user resumes an activity from the Paused state
- `onResume()` is called every time your activity comes into the foreground, including when it's created for the first time
- Implement `onResume()`:
 - to initialize components that you release during `onPause()`
 - to perform any other initializations that must occur each time the activity enters the Resumed state

The slide is presented in a video player interface, with a Windows taskbar visible at the bottom and a small video feed of the professor in the bottom right corner.

So when we resume an activity, `on resume` is called when a user resumes an activity from the paused state. `On resume` is called every time your activity comes into the foreground, including when it is created for the first time. As you saw in the diagram and let me go back to it again.

(Refer Slide Time: 1:11)



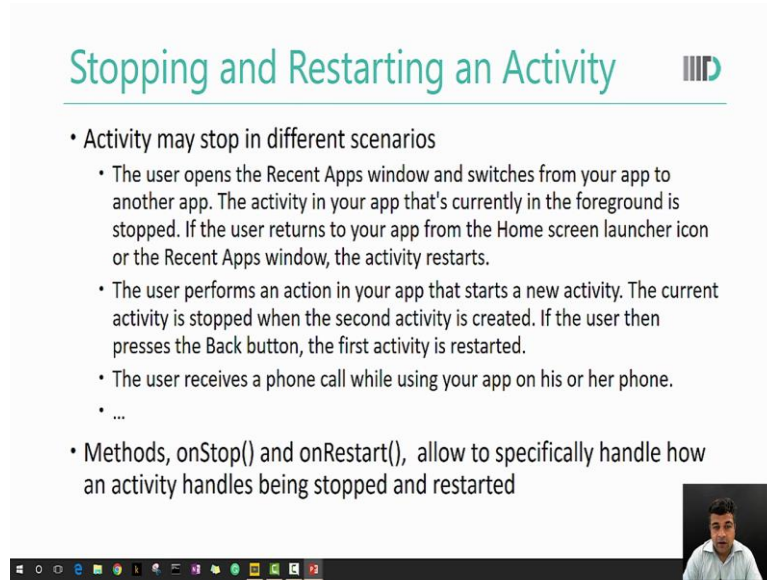
Resuming an Activity

- `onResume()` is called when a user resumes an activity from the Paused state
- `onResume()` is called every time your activity comes into the foreground, including when it's created for the first time
- Implement `onResume()`:
 - to initialize components that you release during `onPause()`
 - to perform any other initializations that must occur each time the activity enters the Resumed state

So On resume is called from the paused as well as from the `(())`(1:16). When you implement on resume you must initialize the components that you release during on pause. So if you release the components when an application went to paused you must reinitialize them as you can understand.


You can also use on resume to perform any other initializations that must occur each time the activity enters the resumed state. Now let's move on to the other two states that is stopping and restarting an activity.

(Refer Slide Time: 1:45)



Stopping and Restarting an Activity

- Activity may stop in different scenarios
 - The user opens the Recent Apps window and switches from your app to another app. The activity in your app that's currently in the foreground is stopped. If the user returns to your app from the Home screen launcher icon or the Recent Apps window, the activity restarts.
 - The user performs an action in your app that starts a new activity. The current activity is stopped when the second activity is created. If the user then presses the Back button, the first activity is restarted.
 - The user receives a phone call while using your app on his or her phone.
 - ...
- Methods, `onStop()` and `onRestart()`, allow to specifically handle how an activity handles being stopped and restarted



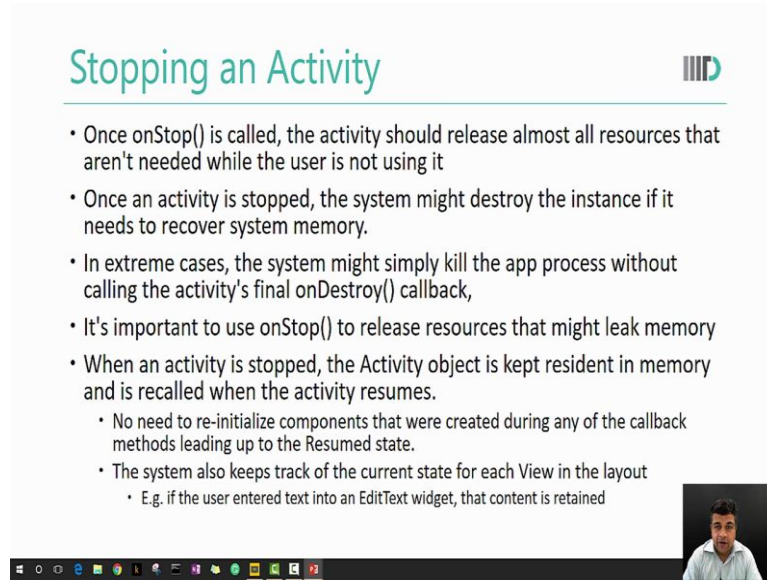
As you may have seen with your use of Android applications and Android phones that an application may stop in different scenarios. For example, from time to time you switch between one application and another that is you go through all your recent apps window and you choose one application over the other application. In this case, the activity in your app that is currently in the foreground is stopped.

And if you return to your app from the home screen launcher icon or the recent apps window, the activity restarts. The many times we do that we press our active application list and we start working on the same application where we are working before. Similarly, if an application starts a new activity in your application. Similarly, if a user performs an action in your app then it starts a new activity.

The current activity is stopped when the second activity is created. If the user then presses the back button, the first activity is restarted. The user receives a phone call while using your app on his or her phone, in this case the phone application starts your activity of your application stops and when you go back your activity on your application starts again.

We use the methods `onStop()` and `onRestart()`, to handle the situations that arise because of this stopping of an application or restart of an application. Let us first see the stopping an activity.

(Refer Slide Time: 3:36)



Stopping an Activity

- Once `onStop()` is called, the activity should release almost all resources that aren't needed while the user is not using it
- Once an activity is stopped, the system might destroy the instance if it needs to recover system memory.
- In extreme cases, the system might simply kill the app process without calling the activity's final `onDestroy()` callback,
- It's important to use `onStop()` to release resources that might leak memory
- When an activity is stopped, the Activity object is kept resident in memory and is recalled when the activity resumes.
 - No need to re-initialize components that were created during any of the callback methods leading up to the Resumed state.
 - The system also keeps track of the current state for each View in the layout
 - E.g. if the user entered text into an EditText widget, that content is retained

So, once on stop is called, the activity should release almost all resources that are not needed while the user is not using it. This is an essential that because on stop is just one step about on destroy that is the destroy method. So whenever on stop is called, you may safely assume that you have made simply leaving an application and your application may need to destroy which means that you must release almost all resources that the user has been using.

And once an activity is stopped, the system actually might destroy the instance if it needs to recover some system memory. That is may be your application or activity does not want to destroy. He wanted to keep it open but your activity list into the stopped state and suddenly because of some other application the system requires more memory.

In this case the system will kill applications which are in stopped state, that may be your application. How the system restores such applications we will study very soon. In some cases system may simply kill the application process without even calling on destroy method. All this indicates to our first point that is whenever the on stop is called, we should try to release all the resources that we have been holding.

When an activity (is) is stopped, the activity object is kept resident in memory and is recalled when the activity resumes. There is no need to reinitialize elements that were created during any of the callback methods leading up to the


resumed state. Please pay attention here, I am saying components that were created during any of the callback methods leading upto the resumed state.

I am not talking after that. The system also keeps track of the current state for each view in the layout. The views were different views objects like buttons adding text, radio button, etc. So if a user has entered a text into an edit text widget, then that content is retained. You need not to worry about.

(Refer Slide Time: 6:15)

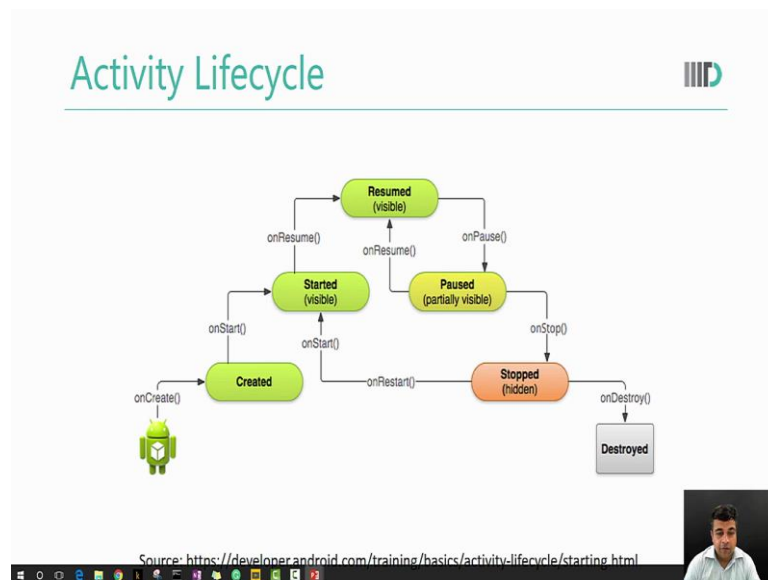
Start/Restart Your Activity

- `onRestart()` is called when activity comes back to the foreground from the stopped state
 - Can be used to perform special restoration work that might be necessary only if the activity was previously stopped, but not destroyed.
- `onStart()` method is called every time an activity becomes visible



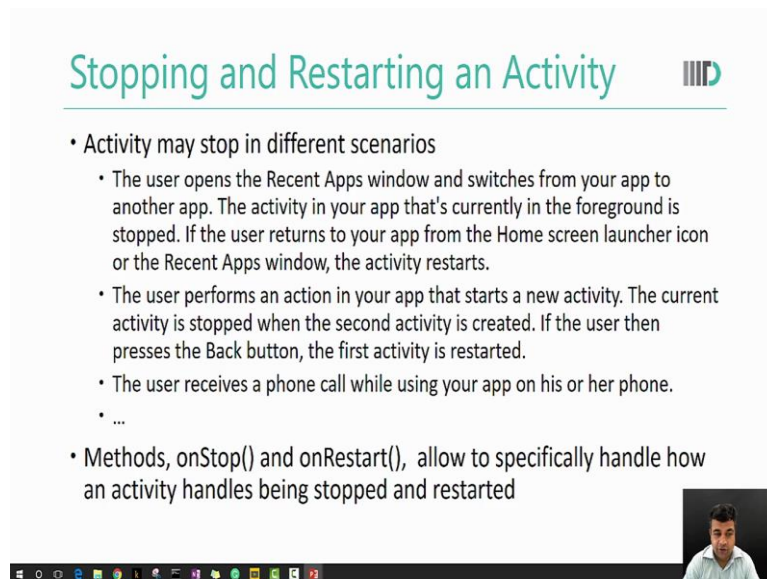
On restart is called when activity comes back to the foreground from the stopped state. Let us see the diagram, here you see

(Refer Slide Time: 6:25)



On restart is being called when activity comes to the start state from the stop. And as you will see how to that after on restart on start is called.

(Refer Slide Time: 6:40)

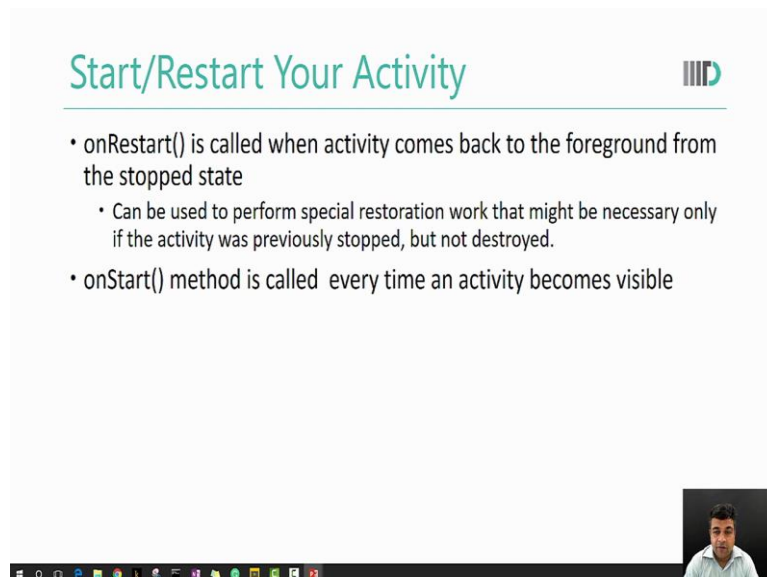


Stopping and Restarting an Activity

- Activity may stop in different scenarios
 - The user opens the Recent Apps window and switches from your app to another app. The activity in your app that's currently in the foreground is stopped. If the user returns to your app from the Home screen launcher icon or the Recent Apps window, the activity restarts.
 - The user performs an action in your app that starts a new activity. The current activity is stopped when the second activity is created. If the user then presses the Back button, the first activity is restarted.
 - The user receives a phone call while using your app on his or her phone.
 - ...
- Methods, `onStop()` and `onRestart()`, allow to specifically handle how an activity handles being stopped and restarted

So on start can be used to perform some restoration while that might be necessary only if the activity was previously stopped, but not destroyed.

(Refer Slide Time: 6:51)



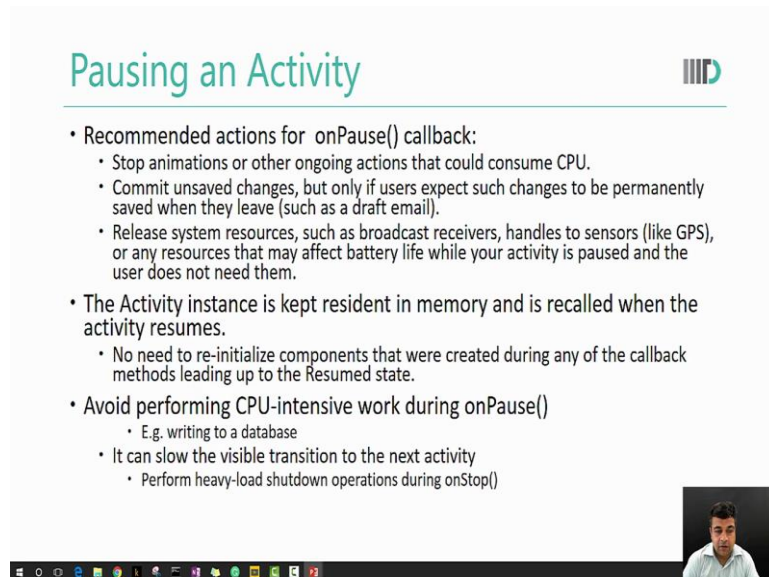
Start/Restart Your Activity

- `onRestart()` is called when activity comes back to the foreground from the stopped state
 - Can be used to perform special restoration work that might be necessary only if the activity was previously stopped, but not destroyed.
- `onStart()` method is called every time an activity becomes visible

And on start will be called every time an activity becomes visible. Which means that if your activity is only going from a visible mode to hidden mode, hidden mode to visible mode on start method will be called every time. While on restart will be called only when the activity went into the stopped state. Please try to

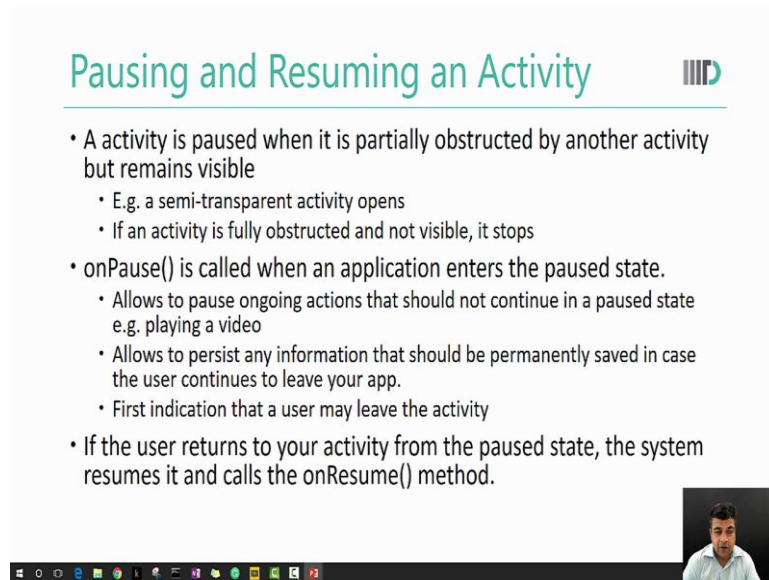
understand the difference in which these two methods are called. Let us see through the diagram

(Refer Slide Time: 7:23)



Pausing an Activity

- Recommended actions for onPause() callback:
 - Stop animations or other ongoing actions that could consume CPU.
 - Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email).
 - Release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them.
- The Activity instance is kept resident in memory and is recalled when the activity resumes.
 - No need to re-initialize components that were created during any of the callback methods leading up to the Resumed state.
- Avoid performing CPU-intensive work during onPause()
 - E.g. writing to a database
 - It can slow the visible transition to the next activity
 - Perform heavy-load shutdown operations during onStop()

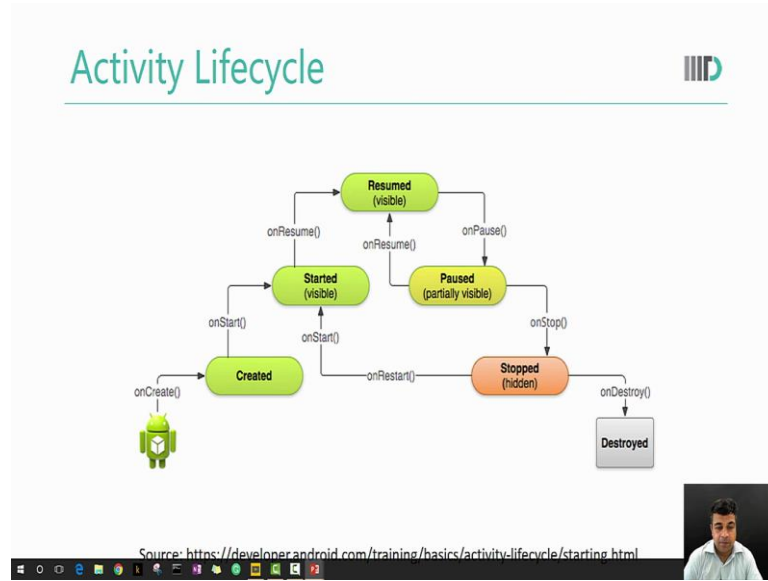


Pausing and Resuming an Activity

- A activity is paused when it is partially obstructed by another activity but remains visible
 - E.g. a semi-transparent activity opens
 - If an activity is fully obstructed and not visible, it stops
- onPause() is called when an application enters the paused state.
 - Allows to pause ongoing actions that should not continue in a paused state e.g. playing a video
 - Allows to persist any information that should be permanently saved in case the user continues to leave your app.
 - First indication that a user may leave the activity
- If the user returns to your activity from the paused state, the system resumes it and calls the onResume() method.

So you see that on start is being called even in beginning because the application has to become visible.

(Refer Slide Time: 7:28)



While there is no on restart, but on restart is called an application is stopped and after that it will call on start to make it visible. Now let us look at we creating an activity.

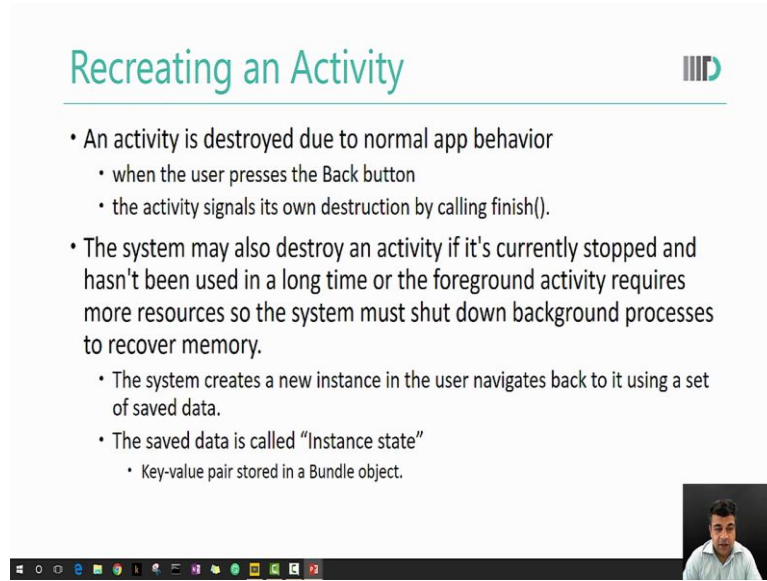
(Refer Slide Time: 7:48)

Stopping and Restarting an Activity

- Activity may stop in different scenarios
 - The user opens the Recent Apps window and switches from your app to another app. The activity in your app that's currently in the foreground is stopped. If the user returns to your app from the Home screen launcher icon or the Recent Apps window, the activity restarts.
 - The user performs an action in your app that starts a new activity. The current activity is stopped when the second activity is created. If the user then presses the Back button, the first activity is restarted.
 - The user receives a phone call while using your app on his or her phone.
 - ...
- Methods, `onStop()` and `onRestart()`, allow to specifically handle how an activity handles being stopped and restarted

Just some time back I told you the scenarios in which an application can be stopped.

(Refer Slide Time: 7:52)



The slide is titled "Recreating an Activity" in a teal font. It features a list of bullet points explaining when an activity is destroyed and how it is recreated. A small video inset of a person is visible in the bottom right corner of the slide area.

- An activity is destroyed due to normal app behavior
 - when the user presses the Back button
 - the activity signals its own destruction by calling finish().
- The system may also destroy an activity if it's currently stopped and hasn't been used in a long time or the foreground activity requires more resources so the system must shut down background processes to recover memory.
 - The system creates a new instance in the user navigates back to it using a set of saved data.
 - The saved data is called "Instance state"
 - Key-value pair stored in a Bundle object.

Now let us see when an application can be destroyed due to normal app behaviour. For example you may press the back button to destroy your application. Your application itself may signal its own destruction by calling finish. That is your application ends. This is similar to any other program that you have written in JAVA.

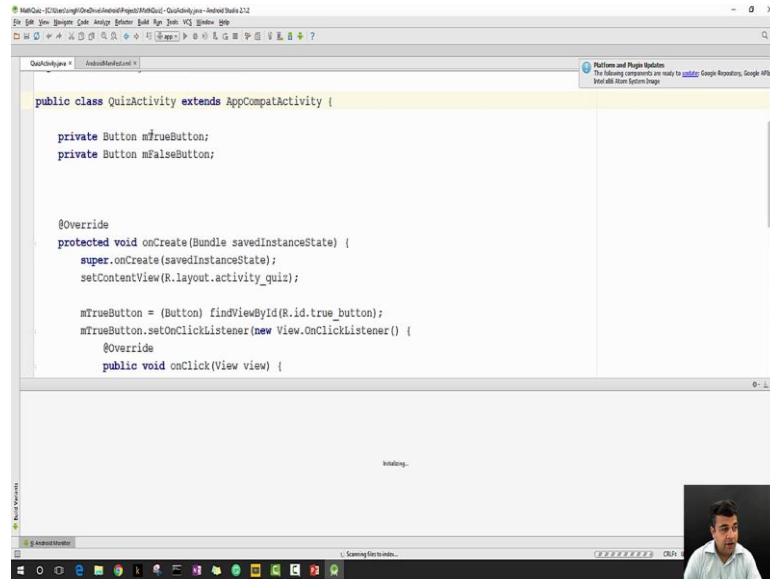
That ends after some time of the executions, these are the scenarios where, which are considered the normal app behaviour. However the system may also destroy an activity if it is currently in stopped state and it has not been used in a long time and the foreground activity requires more resources. Whenever you are running multiple applications on android and if the android system feels that the application which is running required more resources.

It may kill any activity that is currently in the stopped state and has not been used for a long time. However because it is the android system which is killing the application, before destroying the application it creates a new instance in case the user comes back to that application and this new instance is used when the user tries to come back by pressing the back button.

The data which is saved to create this new instance is often called as the instances state which is nothing but a key value pair stored in a bundle object. You may remember bundle object from the parameter that will passed in the on create function. Let me show it to you one more time. I will run my android studio. We

will go back to our program where we will see the bundle object. Again I am using a simpler version of the program not the version that we just created in last week classes. Yes

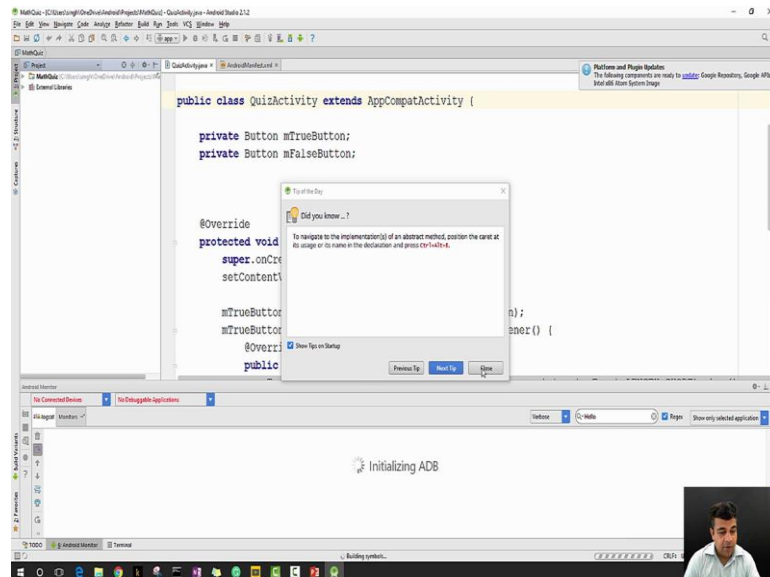
(Refer Slide Time: 10:03)



```
public class QuizActivity extends AppCompatActivity {  
  
    private Button mTrueButton;  
    private Button mFalseButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
  
        mTrueButton = (Button) findViewById(R.id.true_button);  
        mTrueButton.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View view) {
```

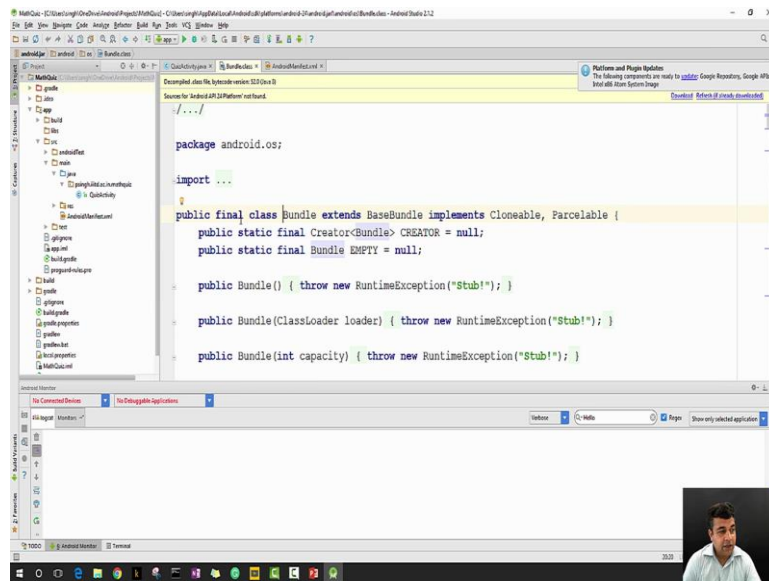
As you seen the on create we are passing a bundle.

(Refer Slide Time: 10:09)



```
public class QuizActivity extends AppCompatActivity {  
  
    private Button mTrueButton;  
    private Button mFalseButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
  
        mTrueButton = (Button) findViewById(R.id.true_button);  
        mTrueButton.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View view) {
```

Did you know...?
To navigate to the implementation(s) of an abstract method, position the caret at its signature or its name in the declaration and press Ctrl+Click.



This is the bundle that we are talking about. The bundle is nothing but an object of type class bundle. If I click I go and I see this final class bundle, next time base bundle implements (()) (10:24). So saved instance state is the object of the type class bundle.

(Refer Slide Time: 10:34)

Recreating an Activity

- The activity is destroyed and recreated each time the user rotates the screen.
- By default, the system uses the Bundle instance state to save information about each View object automatically
- Other state information, e.g. member variables, need to be saved by the app
 - override the onSaveInstanceState() callback method
 - The system calls this method when the user is leaving your activity and passes it the Bundle object that will be saved in the event that your activity is destroyed unexpectedly.
 - If the system must recreate the activity instance later, it passes the same Bundle object to both the onRestoreInstanceState() and onCreate() methods.

The activity is destroyed and restarted each time the user rotates the screen. We saw that in last week's lecture. That when we rotated the screen all the question order was reset. Now by default the system uses the bundle instance state save information about each view object automatically. That is system is handling for all the view objects that you are using.

Anything more than the view object you have to handle yourself. So suppose if you if we were displaying multiple questions 1, 2, 3, 4 which are being monitored by progress of the index in the array. The integer variable that I am using for the array needs to be maintained by me as a developer. While the view object that is the buttons, the texts, edit text, text view fills they will be maintained by the system.

So all the other state information for example number variables, need to be saved by the app by you. You will have to code for it. How do you do that? You override the on save Instance State callback method. The system calls this method when the user is leaving your activity and passes it the bundle object that will be saved in the event that your activity is destroyed unexpectedly.

If the system must create the activity instance later, it passes the same bundle object to both the on store on restore Instance State and on create methods. Let us see this in a simple diagram. From destroyed on save Instance State method is called application received.

(Refer Slide Time: 12:29)

The slide is titled "Recreating an Activity" and features a diagram of the Android activity lifecycle. The lifecycle is shown as a sequence of states: **Destroyed** (orange box) leads to **Created** (green box) via the `onCreate()` method (labeled 2). **Created** leads to **Resumed (visible)** (green box) via the `onRestoreInstanceState()` method (labeled 3). **Resumed (visible)** leads to **Destroyed** via the `onSaveInstanceState()` method (labeled 1). A small Android robot icon is positioned between the **Created** and **Resumed (visible)** states.

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

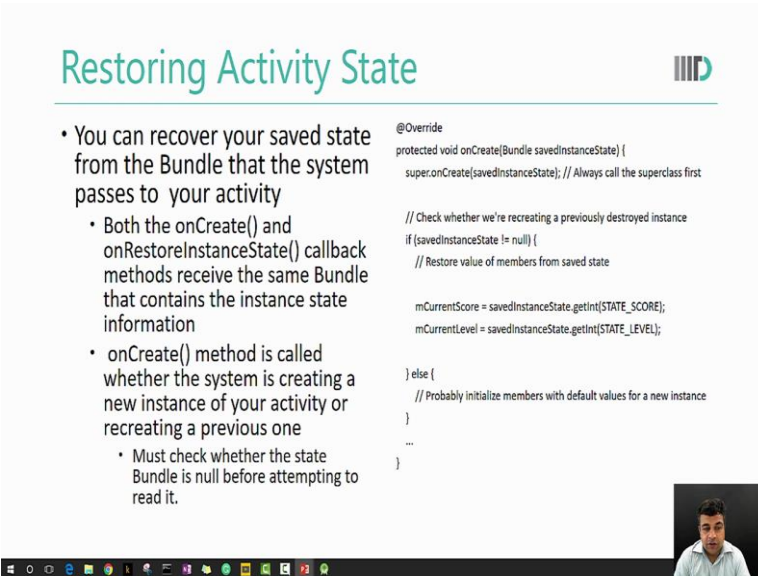
    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
}
```

Source: <https://developer.android.com/training/basics/activity-lifecycle/recreating.html>

From resumed on restore instance state is called an application created. Here I am giving you a simple program sample that shows you how to store values in the bundle and for (())(12:46) key value pair. For example I want to store the values of the state is called and state level. I going to the method I have over written it. The override you see.

In that method I use save Instance State, put int STATE SCORE, m CurrentScore. So mCurrentScore here is my variable that holds the current value of this score and these values are now stored. In next slide I will see how to restore from these values. So restoring (your app) your activity state.

(Refer Slide Time: 13:34)



Restoring Activity State

- You can recover your saved state from the Bundle that the system passes to your activity
 - Both the onCreate() and onRestoreInstanceState() callback methods receive the same Bundle that contains the instance state information
 - onCreate() method is called whether the system is creating a new instance of your activity or recreating a previous one
 - Must check whether the state Bundle is null before attempting to read it.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state

        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    ...
}
```

You can recover your saved state from the bundle that the system passes to your activity. Both the on create and on restore Instance State callback methods receive the same bundle that contains the instance state information on create method is called whether the system is creating a new instance of your activity or recreating a previous one.

Because it can be called both instances, you must check whether the state bundle is null before you try to read it. So let us see a simple program code on how to do it. (So w) so in the on create we go, we want to check (whether the) whether these any useful value in the saved instance state or not? First we check it for null and then we restore our values which are earlier we stored. There we store them in a m Current Score, m Current Level here we restore.

(Refer Slide Time: 14:33)

Recreating an Activity

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...
```

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

Source: <https://developer.android.com/training/basics/activity-lifecycle/recreating.html>

Restoring Activity State

- You can recover your saved state from the Bundle that the system passes to your activity
 - Both the `onCreate()` and `onRestoreInstanceState()` callback methods receive the same Bundle that contains the instance state information
 - `onCreate()` method is called whether the system is creating a new instance of your activity or recreating a previous one
 - Must check whether the state Bundle is null before attempting to read it.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state

        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    ...
}
```

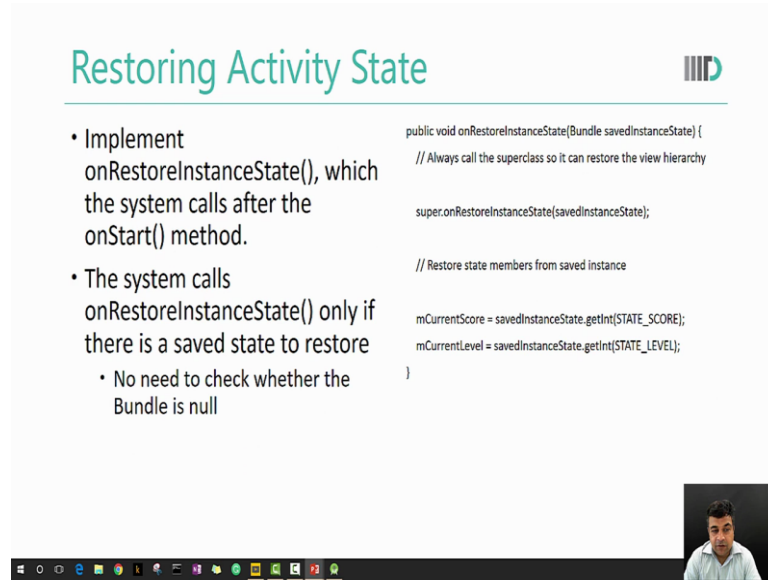
The another way of restoring is to implement the on restore instance state.

(Refer Slide Time: 14:40)

Restoring Activity State

- Implement `onRestoreInstanceState()`, which the system calls after the `onStart()` method.
- The system calls `onRestoreInstanceState()` only if there is a saved state to restore
 - No need to check whether the Bundle is null

```
public void onRestoreInstanceState(Bundle savedInstanceState) {  
    // Always call the superclass so it can restore the view hierarchy  
    super.onRestoreInstanceState(savedInstanceState);  
  
    // Restore state members from saved instance  
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);  
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);  
}
```

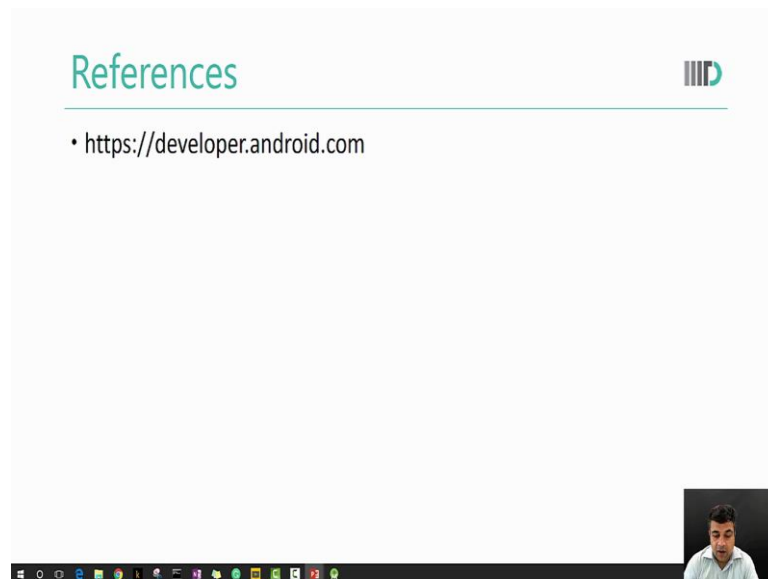


This is a method which the system calls after the on start method. A good point about this method is that system calls it only if there is a saved state to restore which means that there is no need to check whether the bundle is null. So in case, we take this route we define the method and we will directly restore without checking for null or empty. For these lectures I had been using the reference from developer.android.com

(Refer Slide Time: 15:16)

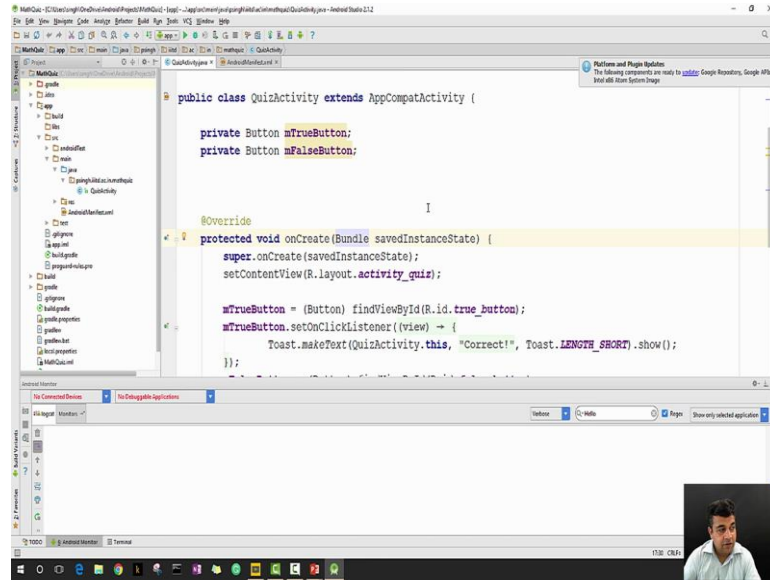
References

- <https://developer.android.com>

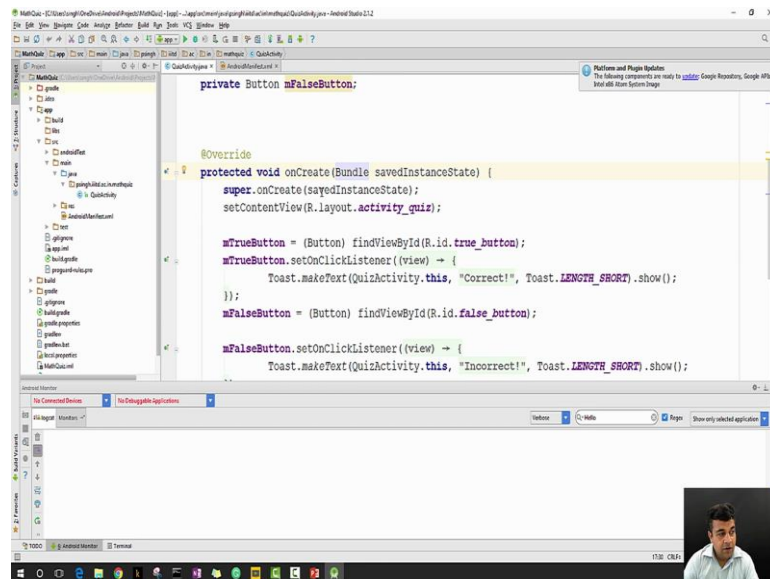


You can go to website and find out information that I am using in this lecture. Now let us go to our program and try to see some of these things.

(Refer Slide Time: 15:33)



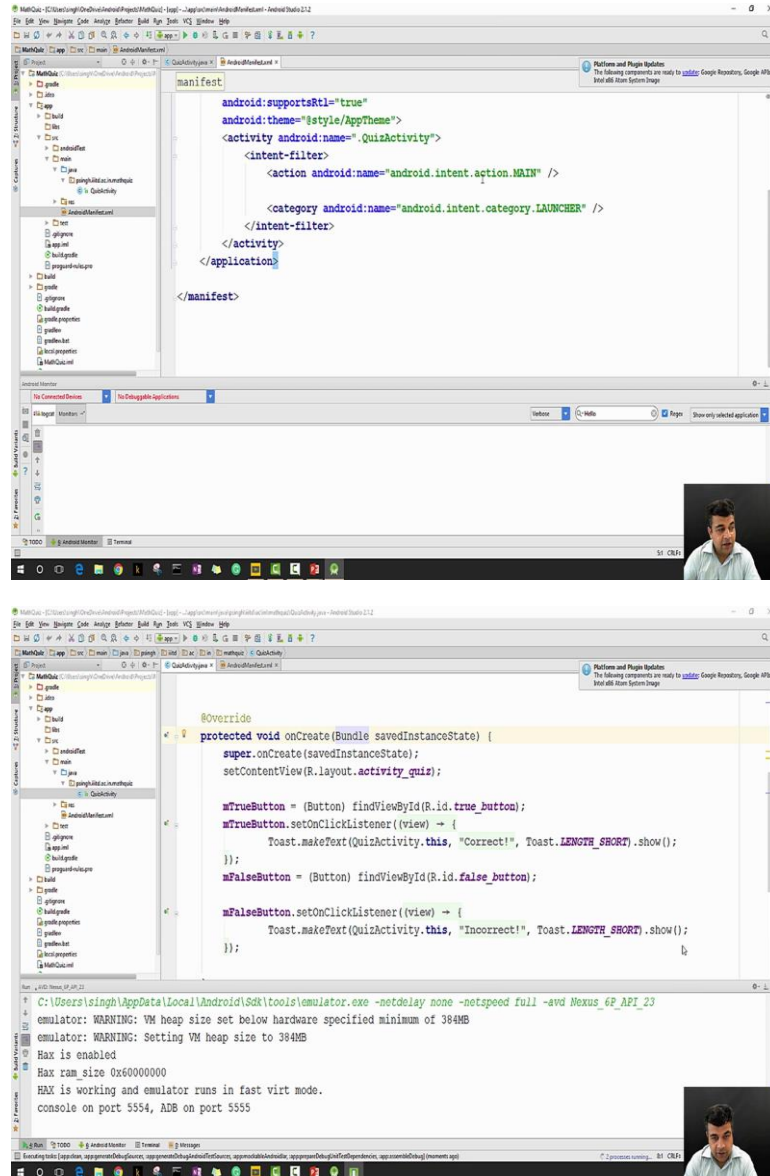
```
public class QuizActivity extends AppCompatActivity {  
  
    private Button mTrueButton;  
    private Button mFalseButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
  
        mTrueButton = (Button) findViewById(R.id.true_button);  
        mTrueButton.setOnClickListener((view) -> {  
            Toast.makeText(QuizActivity.this, "Correct!", Toast.LENGTH_SHORT).show();  
        });  
    }  
}
```



```
private Button mFalseButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
  
        mTrueButton = (Button) findViewById(R.id.true_button);  
        mTrueButton.setOnClickListener((view) -> {  
            Toast.makeText(QuizActivity.this, "Correct!", Toast.LENGTH_SHORT).show();  
        });  
        mFalseButton = (Button) findViewById(R.id.false_button);  
        mFalseButton.setOnClickListener((view) -> {  
            Toast.makeText(QuizActivity.this, "Incorrect!", Toast.LENGTH_SHORT).show();  
        });  
    }  
}
```

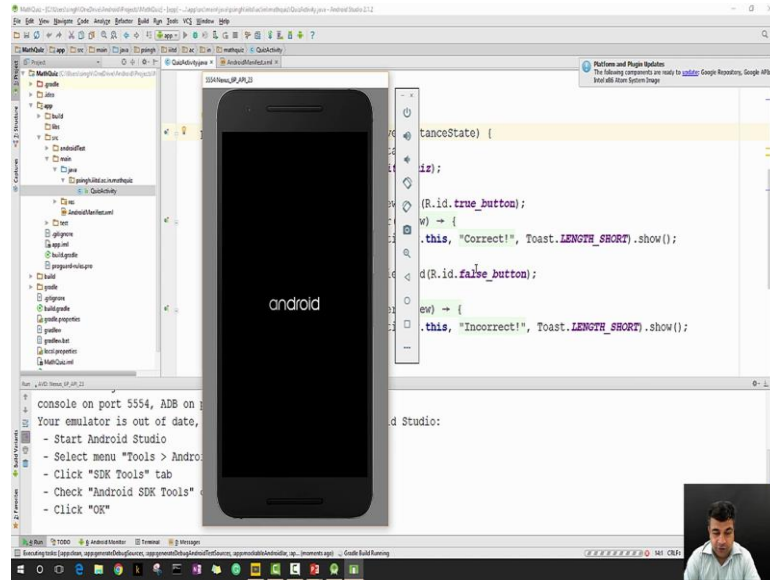
This is our on create method and this is our saved instance state. This is our manifest file which is telling that my quiz activity with the main activity.

(Refer Slide Time: 15:44)



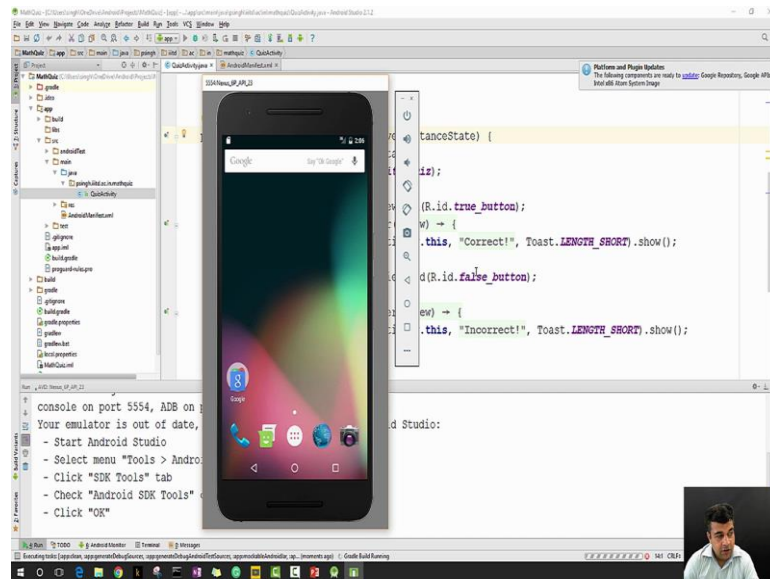
This is the java file. That defines the quiz activity. If I run my program, it will take a while because we are starting the emulator.

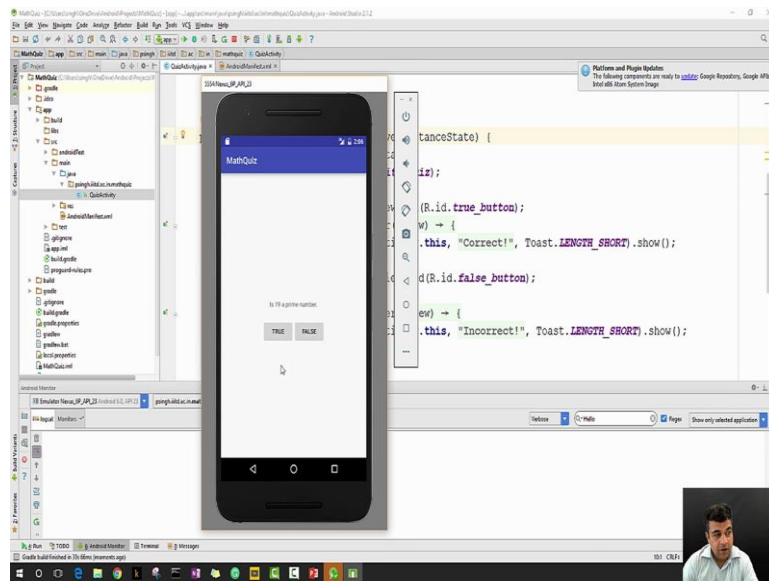
(Refer Slide Time: 16:00)



I am hoping that now we have started the programming in android studio. If you are not then please start. Without writing your own programs, you will not learn android programs.

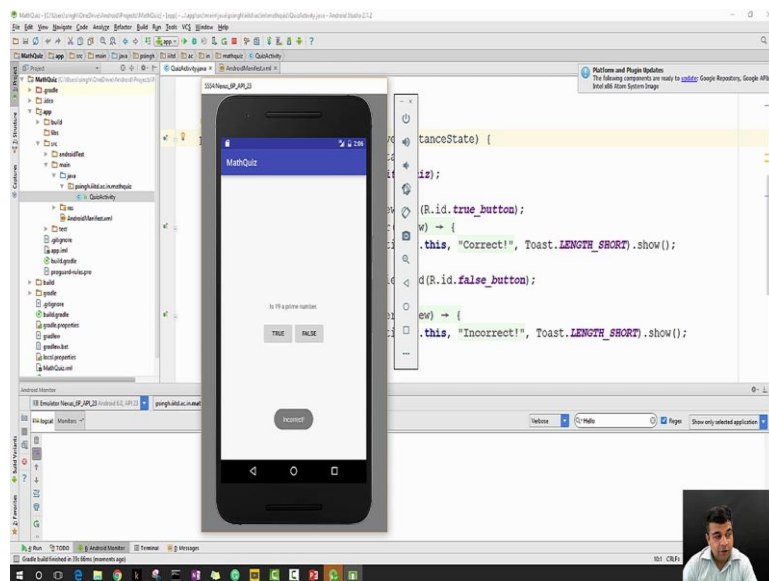
(Refer Slide Time: 16:12)





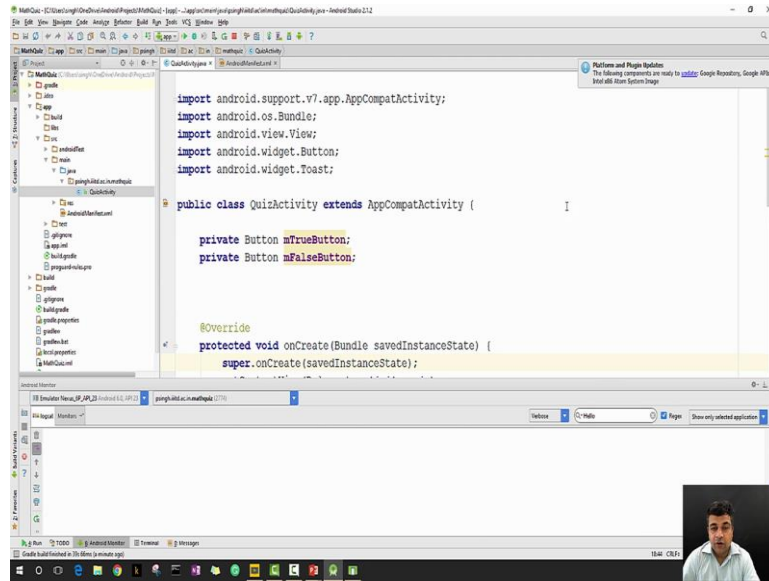
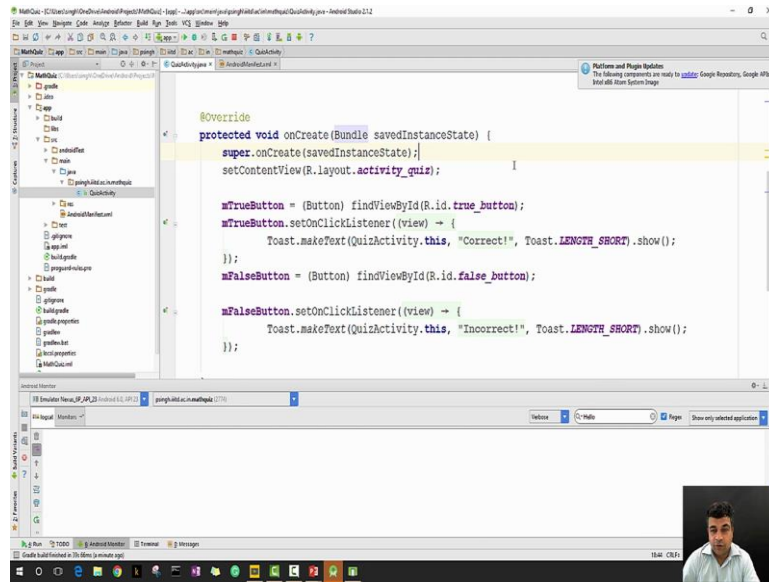
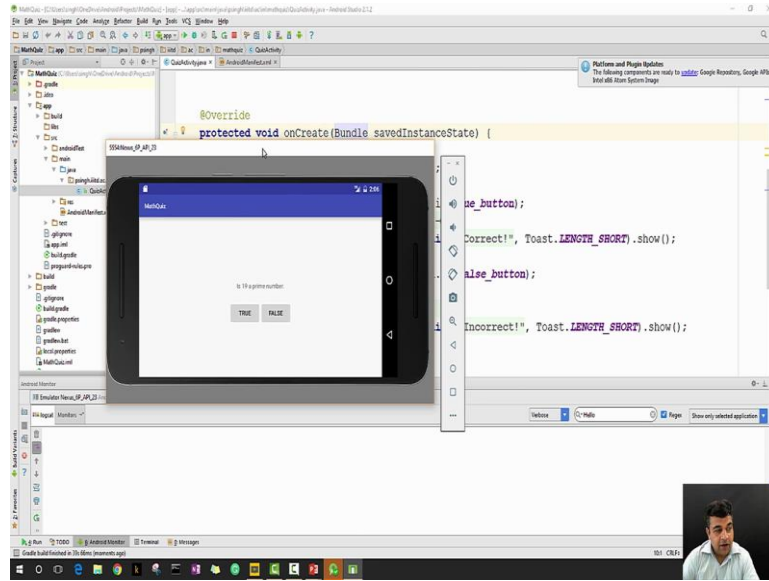
Let us wait in the application as long as to on our emulator. Yes so as you see it is a simplified version of our project.

(Refer Slide Time: 16:38)



Press true, I press false- nothing happens. Let me rotate my phone. Now when I rotated this phone, you cannot see a change but if we had move to another question it would have been restarted from the very beginning.

(Refer Slide Time: 16:47)



Because every time you rotate your screen, the application goes through the complete cycle. And because we are not implementing nor restoring anything save instance state, I will lose the value that I may need in order to maintain current question command.

That is it for this lecture. In the next lecture we will learn how to do logging in android. Logging is a very important feature used to do debugging as well as to understand the behaviour of a of an application. Thank you!