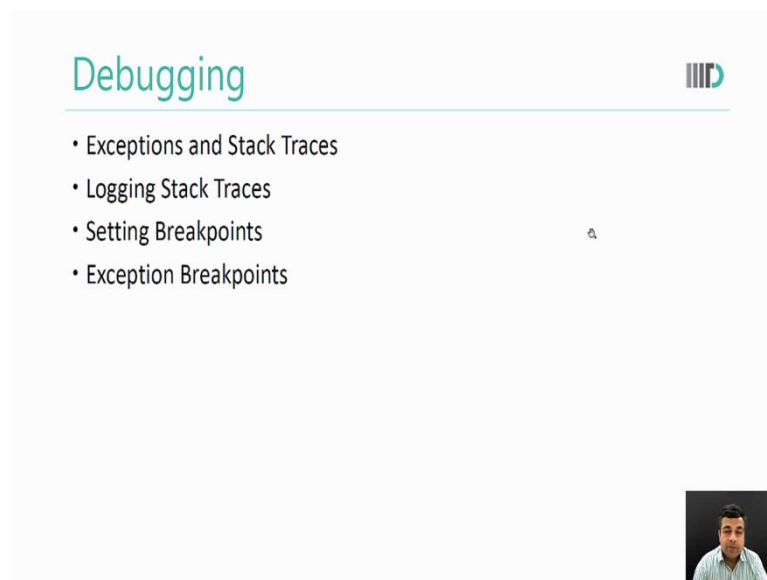


**Mobile Computing**  
**Professor Pushendra Singh**  
**Indraprastha Institute of Information Technology Delhi**  
**Lecture 19**  
**Debugging**

Hello, welcome to your new class to Android Programming. In this class we will learn about Debugging, you may have had a prior experience to debugging, in your prior programming experience, so some of this stuff may look familiar to you and you may want to skip it. However, my advice is to still watch the video and hone your debugging skills. For those of you who have never programmed on Java or who have never used an IDE before, this video is very critical to watch.

What we will learn in the video is that how to debug an Android Application and how to take help of Android Studio in debugging an application. Debugging is a very important skill for programming. If you cannot debug your program effectively, you cannot ever be a very good programmer. This is a skill to master and current day IDEs provide a lot of help to us in debugging our applications. We are going to learn how Android Studio makes it easier to debug Android Applications, so first some basics.

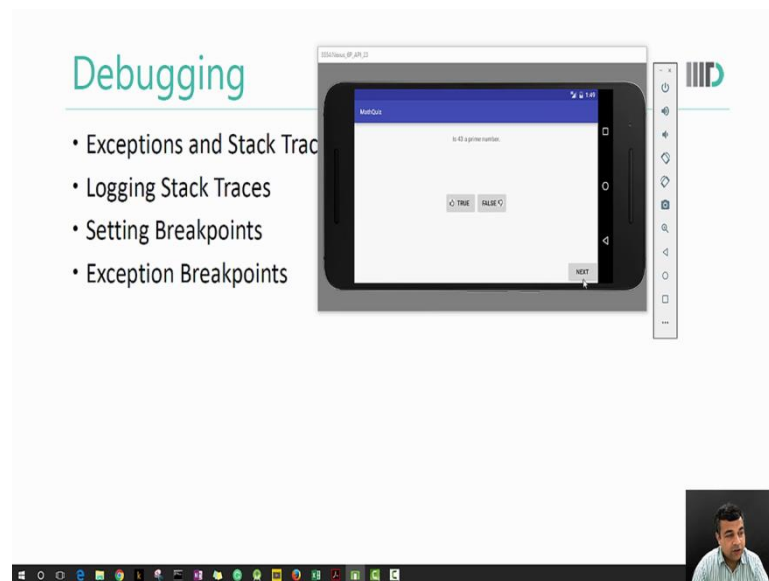
(Refer Slide Time: 1:36)



In Java, we already have a way to control our error behavior that is using the exceptions. Using the exceptions we can either make our program ready to handle errors or we can simply trace it back to the point in the program where the error occur. An important step of that is to maintain the information about the exception and the corresponding Stack Traces. We can log these Stack Traces and then trace it back to the error that we are facing. We can

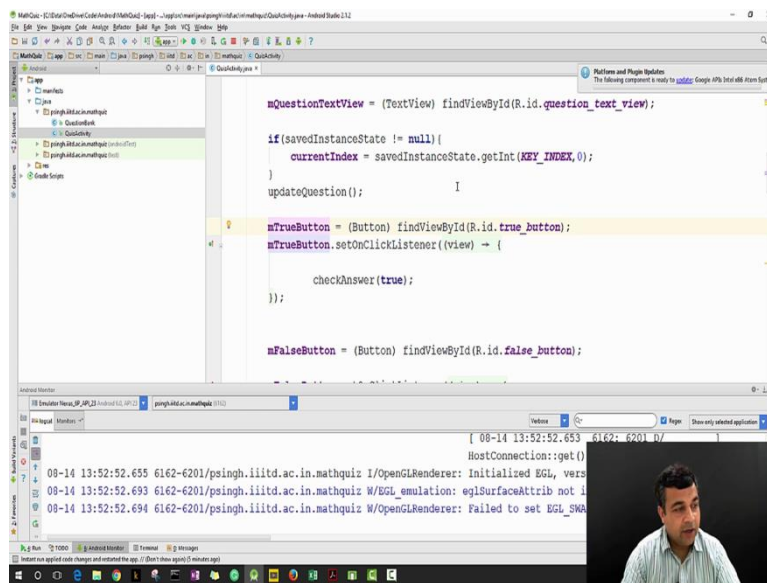
also set Breakpoints in our IDE so that we can enable line by line execution of the code. Android Studio also provides us something called Exception Breakpoints, where we can set the Breakpoints depending on where an exception may occur. Today we will see all of this in action with our Android Studio and with our programming application that we have developed, let us start.

(Refer Slide Time: 2:51)



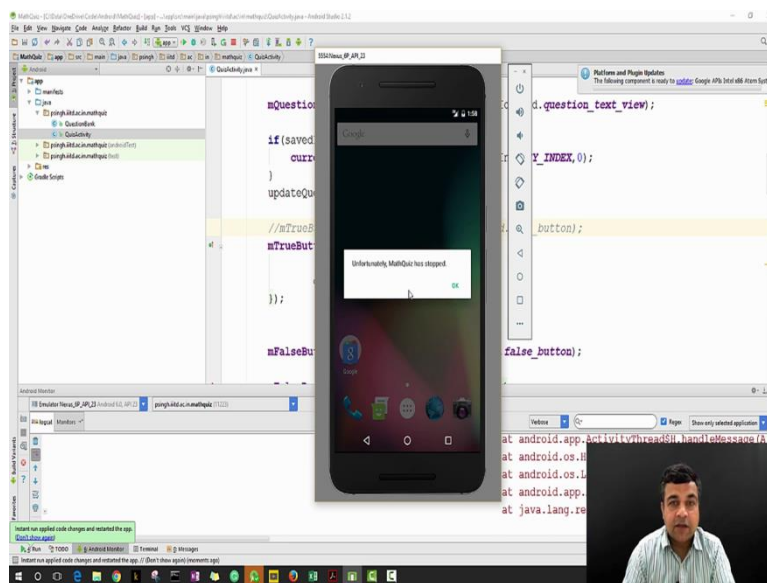
Let me start the Android Studio, as you can already see the Emulator. Let us go back to Math Quiz Application. So this was the application that we have developed and this is also an application which you submitted for your assignment, so you are familiar to it. However, let us run it one more time to see its behavior, next changes to the next question, True or False gives whether the answer given is correct or incorrect. And as you see even if I rotate, now the question does not go back to original, so the application is working perfectly fine. Now let us restart our Android Studio and introduce some bugs. Normally during Application development the bugs appear naturally, because either we oversee them or we unknowingly introduce some things. For example, not writing a correct line, not initializing our variables correctly, etc, and which introduces the bug.

(Refer Slide Time: 4:17)



So let us see since our application was fine, I will introduce a bug deliberately. I am going to comment out this line and now I will execute my application to see what is the impact of commenting out this line. So, let us be ready here and execute our applications, let us see.

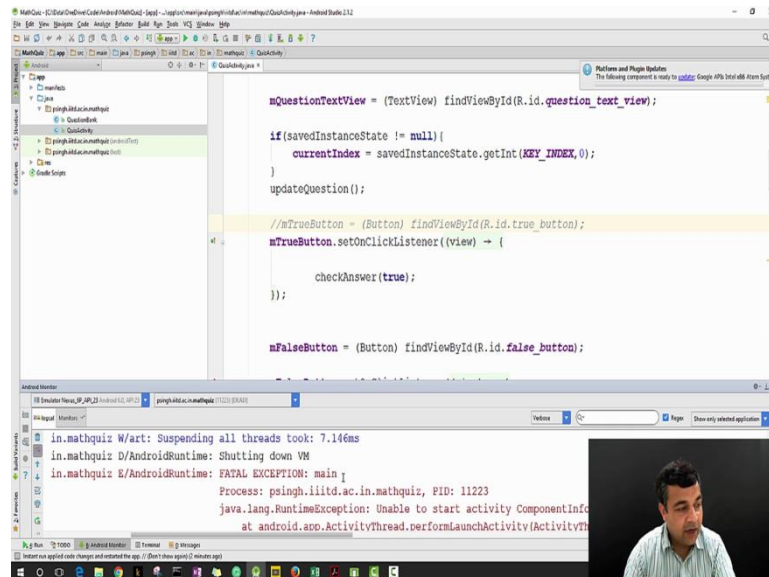
(Refer Slide Time: 4:41)



Oops. So this time our application crashed. This crash was not happening earlier, but because I commented out this line, our application crashed. Sometimes you may have seen similar error on Android applications that we have downloaded from the Play Store. These errors happen because the programmer oversees an error which he should have corrected before releasing the app. The aim of this video lecture is to make sure that you do not make the same

mistake and then you debug your application thoroughly before you release it to the Play Store, so let us go through it.

(Refer Slide Time: 5:34)



If I look at it, I can see that there is some red portion in my Logcat window. The Android Studio, in a very helpful manner helps us create. If I see my Logcat window, if I see my Logcat window I can find that there is some red lines here. This is Android's way of telling us that some exception has occurred. Now let us go through this, and try to find out what happened. So, I hope all of you can concentrate here on the bottom screen. It says “Android Runtime Fatal exception Main”, gives the PID, gives the process and then it tells the exception that has occurred “java.lang.RuntimeException” the brief description is unable to start activity component and then it goes on to the detail At at at different lines.

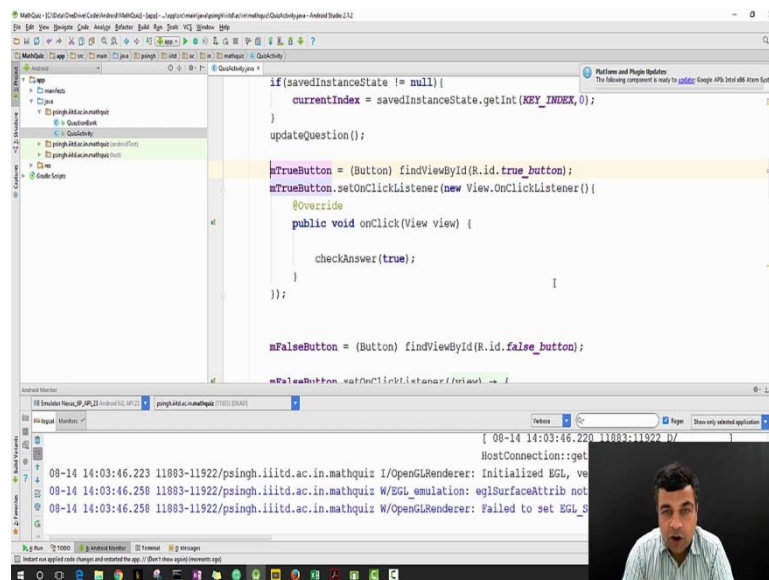
If you have already programmed in Java you should know that this is what we call as Stack Trace. So the Android Studio has dumped all stack on us and we have to find out where the error occurred. A good way to find it out is to keep going through the trace. Now, we have found a line which says “Caused by java.lang.NullPointerException: Attempt to invoke virtual method” gives the name of the method on a NullObjectReference. So what it is trying to tell us is that we tried to run a method on an object which was not initialized, this has caused the NullPointerException, which has caused the ultimate Runtime Exception.

So while debugging you need to keep going down to the Stack Trace till you find a line called caused by and try to redefine it. Now the next line gives us the line number where it has occurred, let me click on it. If I lick on it, it shows me this line. And as you can see that I had

commented this line just before this. But what was in the line that I commented. If you see in that line, I was actually assigning mTrueButton variable the actual object, using the findViewById method. Now because that line was commented, mTrueButton was never initialized and in the next line when I tried to call setOnClickListener on my mTrueButton I got a NullPointerException because mTrueButton is currently null. And invoking on it resulted it into the NullPointerException which then caused Runtime Exception and our app crashed.

So if you are getting an Exception in your Android, keep going through this Stack Trace, find out a line called Caused by, see the next three lines will give you a very good heap. The first line gives me exactly line number where this error is occurring, try to look upwards of that line because that line was, where this pointing was executed, so the error must have happened earlier. In our case, this was easy to find out and we could see that. This is a simple example of what we call Logging Stack Traces. Now let us go back to our Android Studio again and start setting some Breakpoints. First let me clear this error and hopefully if I run my application, my application should run once again perfectly. See, and as you see our application is working fine.

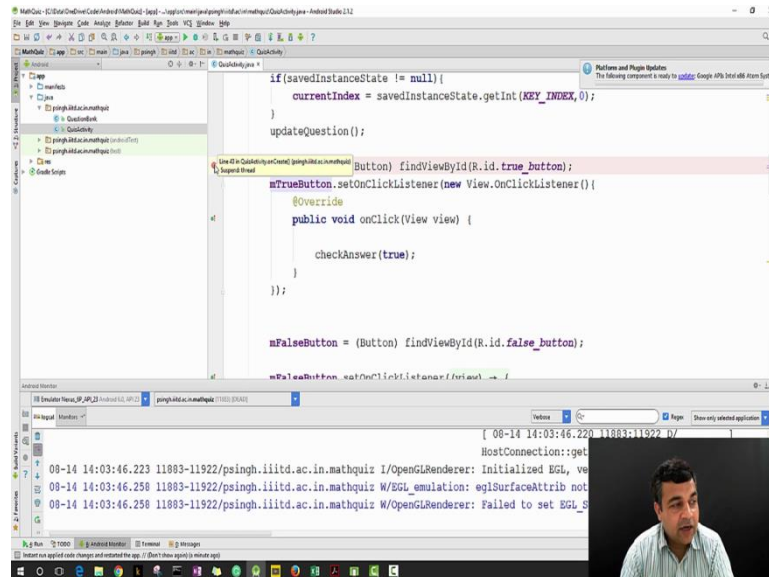
(Refer Slide Time: 10:05)



Let us stop the execution and learn more about Breakpoints. So what is a Breakpoint. A Breakpoint is a point in your program where you want the execution to stop and you want to have a look on how the program has progressed so far. Breakpoint setting in an IDE is a very useful technique for debugging because it allows us to go through execution line by line. Let us see how we can do that in Android Studio. The procedure is very much same as with

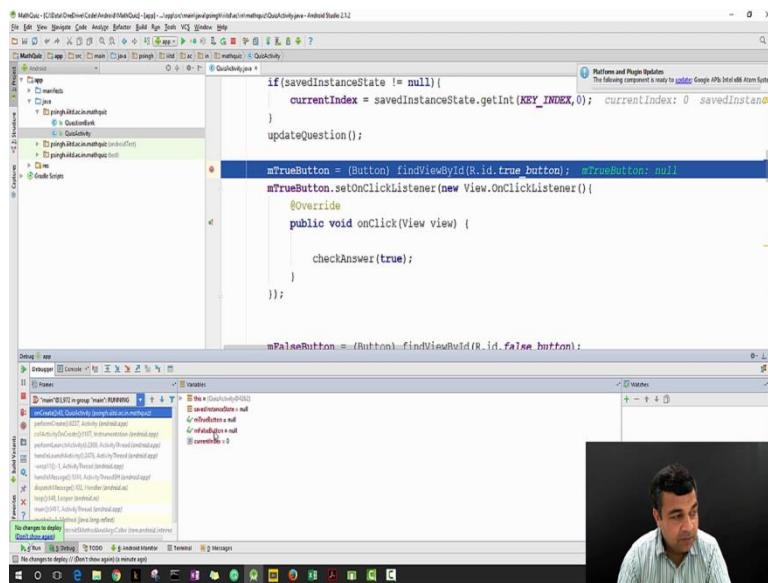
Glimpse or another IDE. So unlike last time where I introduced a bug, this time I will set a BreakPoint here. For setting Breakpoint I have to do nothing, except just click on the Grey Bar besides where I have Witten my code. So as you can see, I could see a BreakPoint here and if I click it again the BreakPoint disappears, if I click once the Breakpoint comes.

(Refer Slide Time: 11:03)



Breakpoint is indicated by a red point and if I point my mouse I can read a message which says that “This Breakpoint is on line 43” and on this point the execution will be suspended. Now in order to see these Breakpoints in action we will have to run our program in Debug mode. The Debug mode is enabled by pressing the button next to the Green Arrow. If you take your mouse to the Green bug button, so this button looks like a bug. It says Debug App. It can also be done by pressing Shift and F9, but let me press the Debug button.

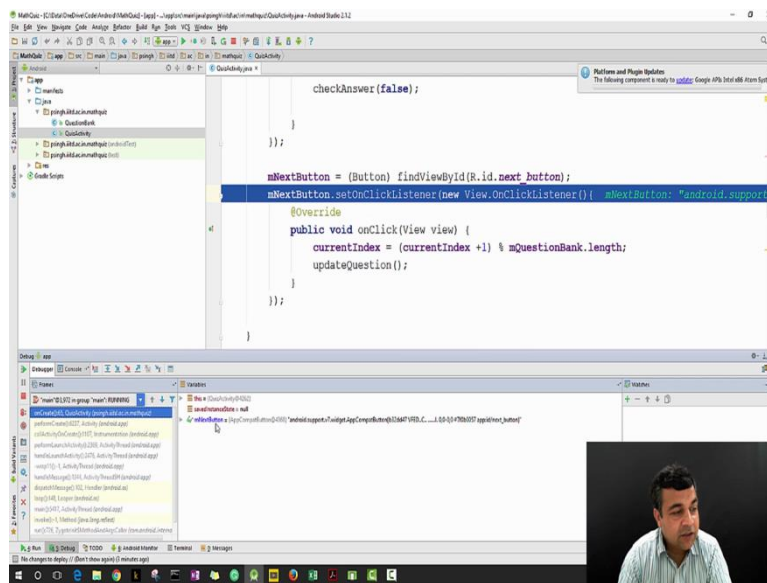
(Refer Slide Time: 12:49)



Debug is similar to execution, except that it gives us more information; it stops at Breakpoints and is a mode which is made entirely for debugging your application. Now when I press the Debug, you can see that my Window pane at the bottom has changed. Earlier I was getting Logcat here, now I am getting Debugger here. Not only that but somehow, my program came and the line where I have put the Breakpoint is now marked as blue on my screen, you may get a different color. What it shows is that now program execution has now stopped at this point. Now this is also allowing me to check the value of different variables till this point. For example, I can see that `currentIndex` so far is 0. My `mTrueButton` is null, my `mFalseButton` is also null.

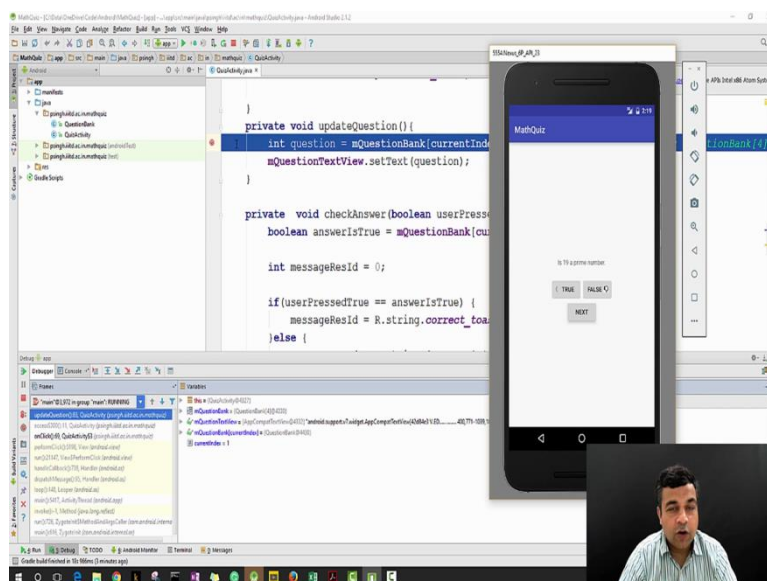
There are some other icons shown here. First icon is StepOver which I can activate using F8. Second icon is StepInto which I can activate using F7 and then there are various other icons. I will mainly making use of StepOver and StepInto. StepOver is used to go over a line without going deeper into it. So suppose I am calling a method and my Breakpoint is set to the point where the method is called, if I press StepInto my execution will move into the method. But if I press StepOver my execution will just move over to the next line. For the time being let us just press StepOver. I have pressed this StepOver, which means that now I have come to the next line. And see the impact. In this line because I initialized my `mTrueButton`, I can now see that among the variables while the `mFalseButton` still remains null, `mTrueButton` has now been initialized.

(Refer Slide Time: 14:48)



Let me press the StepOver again, it comes into mFalseButton which is still showing as null. Press it again and my mFalseButton has also been initialized because I am passed this line. now let me press StepInto. If I press StepInto then it tries to go into the mNextButton and mFalseButton, I come out of it, I do StepOver again, I come to mNextButton which is still null, do StepOver and my mNextButton is also initialized, so my execution is going line by line. Now, I have removed the Breakpoint from its earlier position and I have added a new Breakpoint to my update Question. I am running the Debugger and my application is displayed here. Now you know that updateQuestion is called every time we press the Next button.

(Refer Slide Time: 15:38)

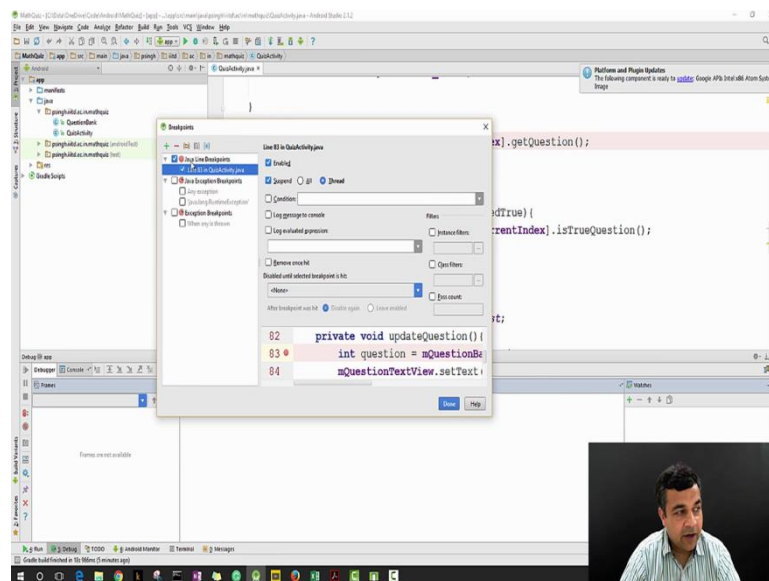




So let us press the next button and see what happens. The moment I pressed the Next button, my question is not changed, because my execution is stopped at the line where I had added the Breakpoint. This is a wonderful opportunity for me to check what is the current state of all the variables that I have defined. So I can go to this window and I can check the value of the variables. For example, currently it is showing that my Current Index is 1, because my Current Index has been changed and now it needs to update the question. In order to move forward, I will press the Resume button, so this is the Resume button. I press the Resume button and as you can see that our question is now changed, our program is in execution again. When I press the next button again, again the execution stops, again I press the Resume and my question changes.

So this shows you correctly, the power of a debugger and a Breakpoint. You can set the Breakpoint and you can control your execution making it to stop at each line if you want it to. Let us now move on to the third topic which is the Exception Breakpoints. First let us stop our debugging, I stopped the debugging and now I will go to the top menu.

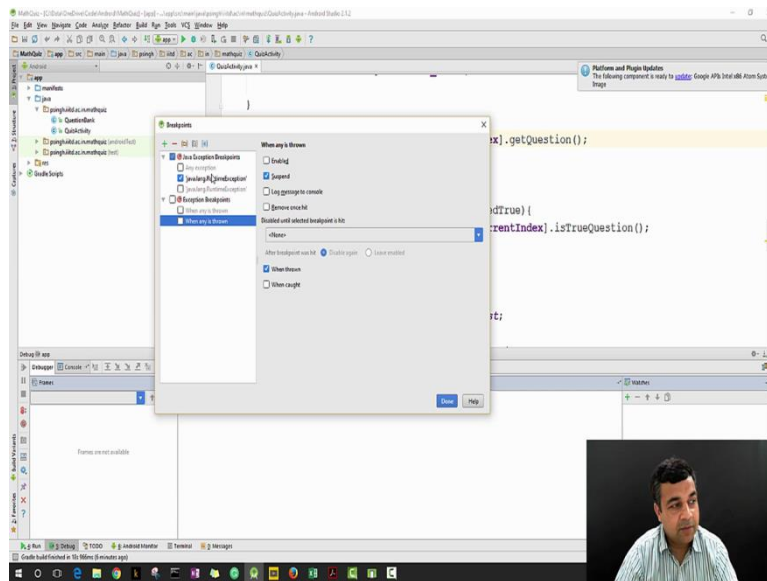
(Refer Slide Time: 17:29)



At the top I have a menu item called Run, inside Run, towards the bottom I have something called ViewBreakpoints. Let me click it. When I click ViewBreakpoints, I see Java Line Breakpoints and I see Java Exception Breakpoints and I see Exception Breakpoints. As you will see that in Java Line Breakpoints it is already showing the Breakpoints that we have added, we can also see that here. Let me first remove this Breakpoint, now I do not have any Java Exception Breakpoints. I have got only Java Exception Breakpoints and Exception Breakpoints.

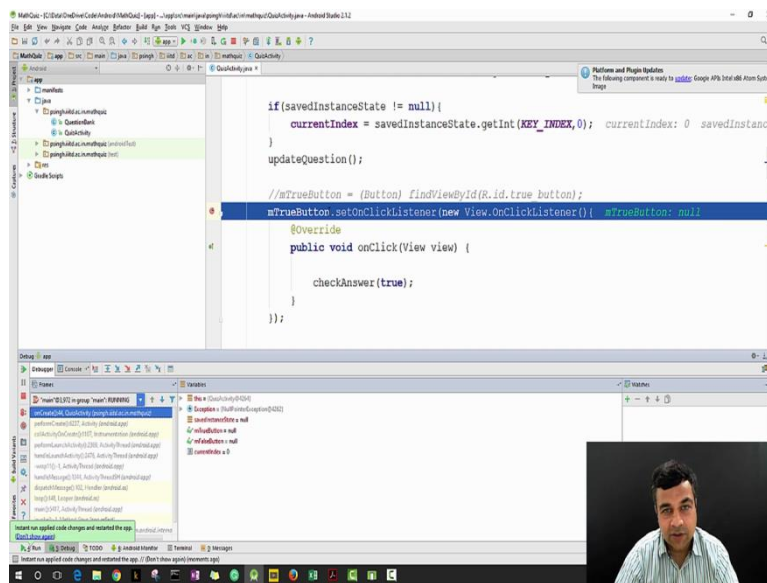
The idea behind Exception Breakpoints in Android Studio is that sometimes I may not know where my exception is being found. That is I may not know where I should add my Breakpoint. So the Java Android Studio helps in doing that. Let me press the + button, among the menu let me check the Exception Breakpoint or let me change, check Java Exception Breakpoint, it will show me a new window and as you know that last time when we have commented out our initialization of True button, it has thrown us as a Runtime Exception. So I will set my exception Breakpoint to Runtime Exception only.

(Refer Slide Time: 19:07)



So I have set that for java.lang.RuntimeException, I am setting a Java Exception Breakpoint. This is a way to say that if in my program anywhere a Runtime Exception is to be thrown please add a Breakpoint just before that line, now let us go back to our code and make sure to introduce a bug that last time resulted to a Runtime Exception. I have done that now let me now Debug my application and as you have seen I have not set any Breakpoints manually.

(Refer Slide Time: 20:03)




Now, let me run my application in Debug mode. My application starts, launches and as you see, that even without adding a Breakpoint my execution stops at the line which last time has resulted into Runtime Exception. This is the beauty of Exception Breakpoints in Android Studio. They take away the need from the programmer to define the right places where to keep Breakpoints. This is a very good feature which you should use frequently in programming, especially if you are getting exceptions and you do not know from where they are being thrown. Using Exception Breakpoints you can find such points. However as a good programmer you should be defining your own Breakpoints and you should work out with that to make your application bug free.

Let me uncomment the part, stop the execution and try to run my application again with the same Exception Breakpoint set and as you know or you may have guessed this time our execution does not stop because there is no place in our program which may throw a Runtime Exception. So this was all about Debugging in a dynamic context that is when your program was running. Whatever we have discussed can simply be used for any Java Program or for any other programming languages because the principles of the debugging remain same.

(Refer Slide Time: 21:38)

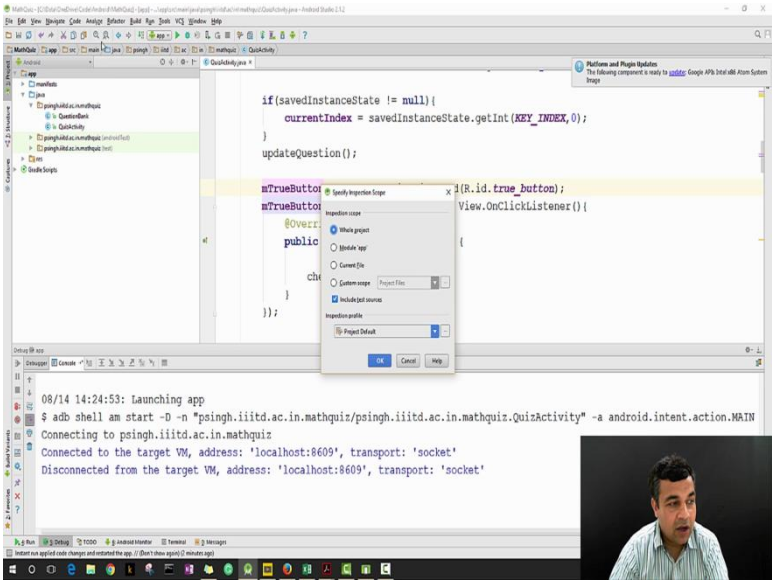
## Android Specific Debugging

- Lint – static analyzer
- Issues with R class
  - Check the validity of XML in your resource files
  - Clean your project
  - Sync your project with Gradle
  - Android Lint



Now we will go into some Android specific debugging, that is provided by Android Studio. One of the techniques that is provided by Android Studio is a static analyzer called Lint. Static analyzer analyses your program statically that is without running the code. Static analysis is a very important branch of Software Engineering, where we try to find out that how a program can execute by just looking at the source code of the program. You may want to know more about the static analysis of programs. Please try to find a suitable book or recourses which tell you that, for our course we will only cover the Lint static analyzer as given by Android Studio.


(Refer Slide Time: 22:42)



The screenshot shows the Android Studio IDE. The main editor displays Java code for a button click listener. A dialog box titled "Specify Inspection Scope" is open, showing options for inspection scope (Whole project, Module, Class, Custom scope) and whether to include generated source. The terminal window at the bottom shows the command to launch the app and the resulting output, including connection and disconnection messages to the target VM.

```
if (savedInstanceState != null) {  
    currentIndex = savedInstanceState.getInt(KEY_INDEX, 0);  
}  
updateQuestion();  
  
trueButton.setOnClickListener(  
    new View.OnClickListener() {  
        @Override  
        public void onClick(  
            View view) {  
                trueButton.setText("True");  
            }  
    });  
};
```

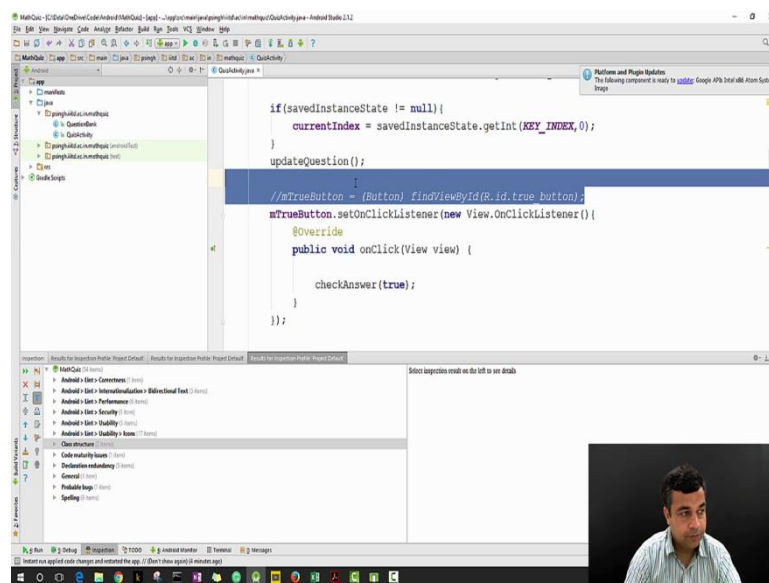
```
08/14 14:24:53: Launching app  
$ adb shell am start -D -n "psingh.iiitd.ac.in.mathquiz/psingh.iiitd.ac.in.mathquiz.QuizActivity" -a android.intent.action.MAIN  
Connecting to psingh.iiitd.ac.in.mathquiz  
Connected to the target VM, address: 'localhost:8609', transport: 'socket'  
Disconnected from the target VM, address: 'localhost:8609', transport: 'socket'
```



Let me open the Android Studio again, and let us go to something called Analyze. In the top menu you will see this item called Analyze and the first item is Inspect Code. Let me press the Inspect Code. It further asks me what do I want to inspect, so I am choosing whole project, let me press the Ok button. When I press the Ok button my static analyzer starts running and I see following result. As you see, it gives me result on multiple dimensions, performance, security, usability. For example, it is even figuring out small design considerations such as the button should be borderless.

So how does the Lint do it? Well, the Lint has a very good understanding of the Android studio and how we should write Android program. It uses that knowledge to analyze your program code and flags you some warnings. Lint is a very good tool to look when you are deploying your application to the Play Store and try to take into account the warnings generated by the Lint.

(Refer Slide Time: 24:15)

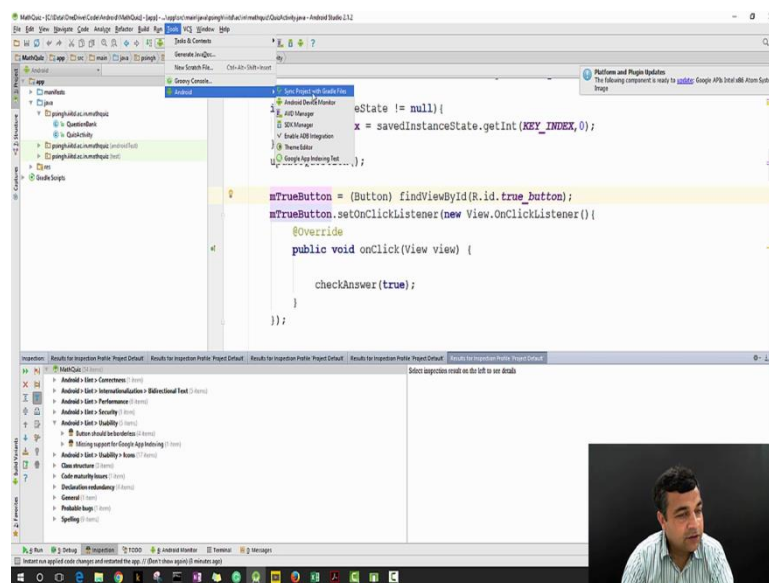


Now let us see what else Lint can help us with. Let me go back to our code and instead of this line, I will comment, let me try to assign it a different Id of say question text view, this is a button, what I am trying to assign an Id of text view, save the program and I run my Lint analyzer. This time you can see that it is giving me an error saying “Mismatched view type”, because it could figure out without even running my code that I am trying to assign a text view id to a button. So these are some other errors in which Lint can help us. Let us make our program right again. But even besides this, Lint is very good in giving us good advice on usability, security, performance, internationalization, correctness etc.

Now, let us go back to our last topic of the Debugging class. Many of you have already faced issues with R.java file and these errors are very very difficult to debug. In fact there is a very good reason why R.java is not made visible to you in the first instance. But still, if you happen to run into errors with your R.java file here are few steps, that you may take which may help you in making your life easier. Number 1, the first step when you are getting an error with R.java is check the validity of your external files. As you know that you assign your resources in the XML file and those resources will have a corresponding resource Id in R.java file. Sometimes it is the validity of your XML file that is causing the error, so the moment you start getting error in R.java go back and check if your XML is valid or not.

Some other techniques that you can use is you may clean the project and rebuild everything from the 0, sometimes it gets rid of the error then you may also Sync your project with Gradle.

(Refer Slide Time: 27:16)



So you may go here, you can go to Run, go to the Build, here is the Clean project which I was talking about and this is the Sync project with Gradle files. And then if nothing works then try to run the Lint, find out if Lint can give you a appropriate warning to solve your R.java error.

(Refer Slide Time: 27:41)



The slide is titled "References" 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 letter 'D'. Below the title, there is a list of 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>

In the bottom right corner of the slide, there is a small video inset showing a man with dark hair and a light-colored shirt speaking.

This was all for your today's lecture. As usual I have used references from Big Nerd Ranch guide and Developer Android websites. Debugging is a very important concept and today we learned how to debug a Java program in general and how to use Android Studio for debugging Android applications. With today's lecture, your basic training as an Android developer is complete. You have created an application with a single activity, you know how to log and find out about what activity lifecycle currently the activity is going through. You know basic debugging techniques, you know basic UI techniques.

Now onward we will move fast into the advanced topics. So far the pace of learning was a bit slower to let you come up with the new programming environment of Android, but now I will assume that you have done your homework, you created an app and you are ready to delve in to the advanced concepts of Android Programming.

Thank you.