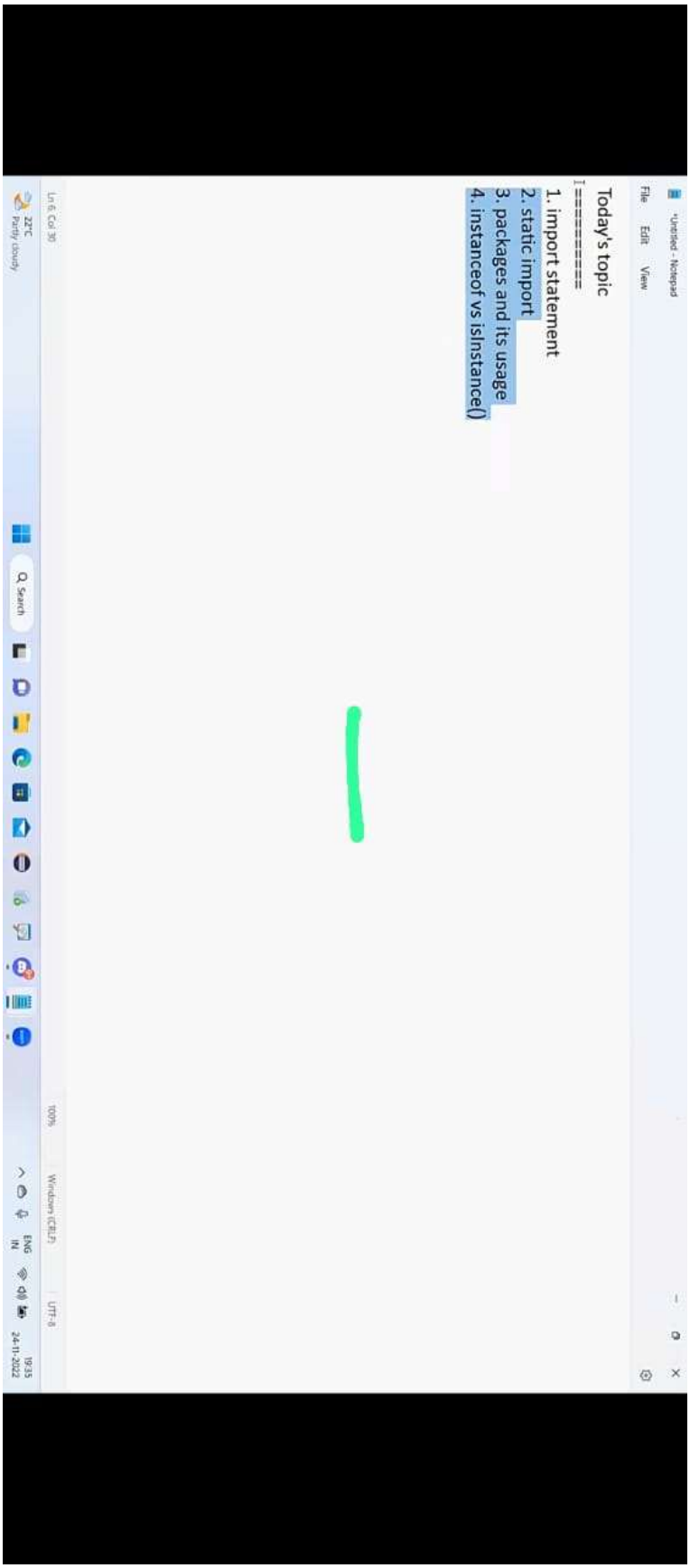
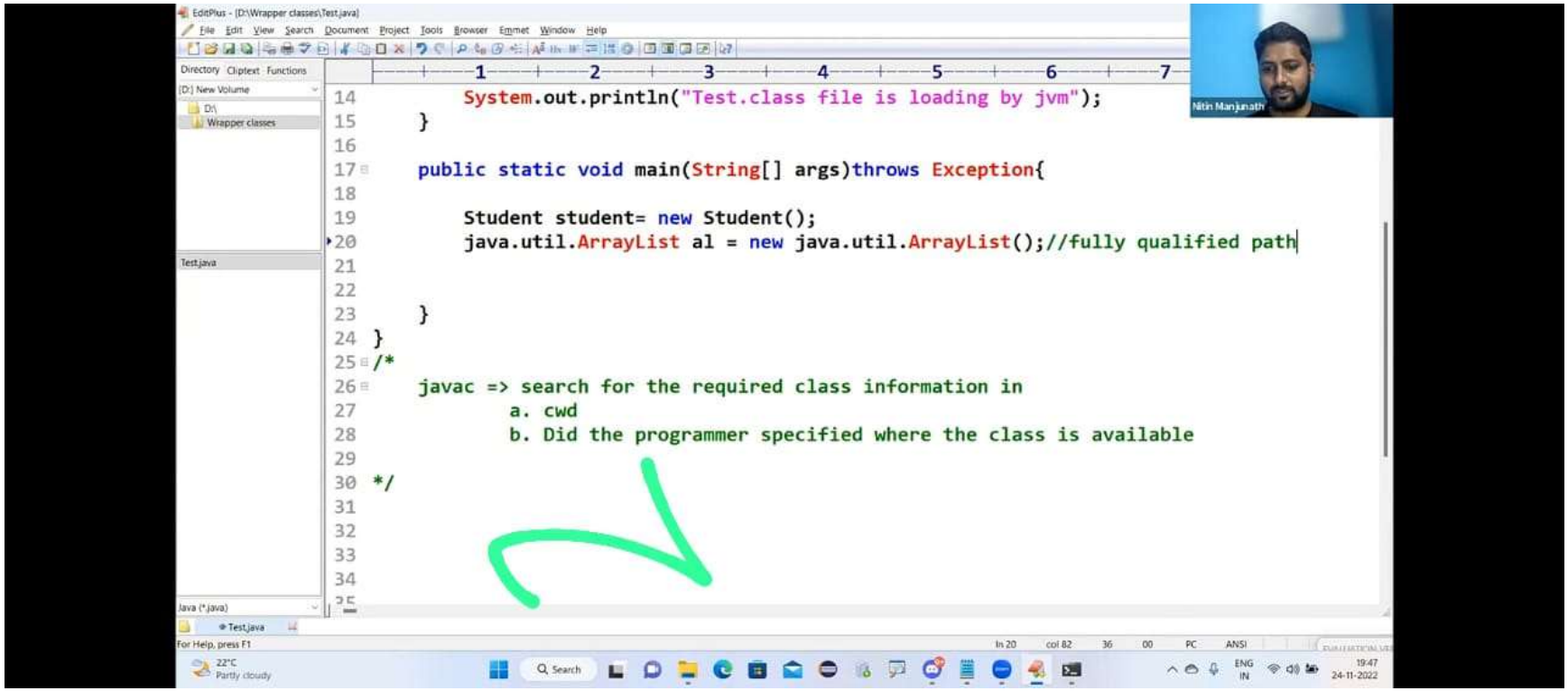
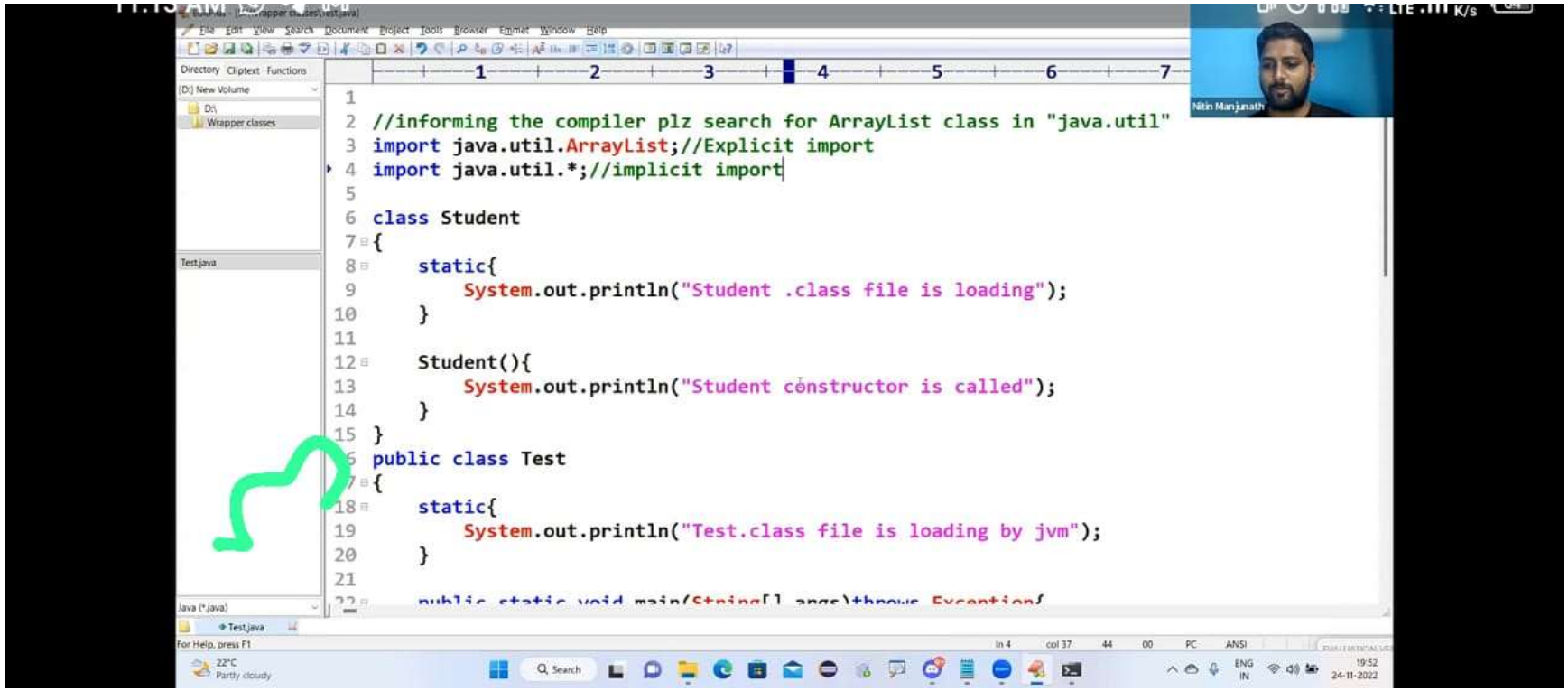
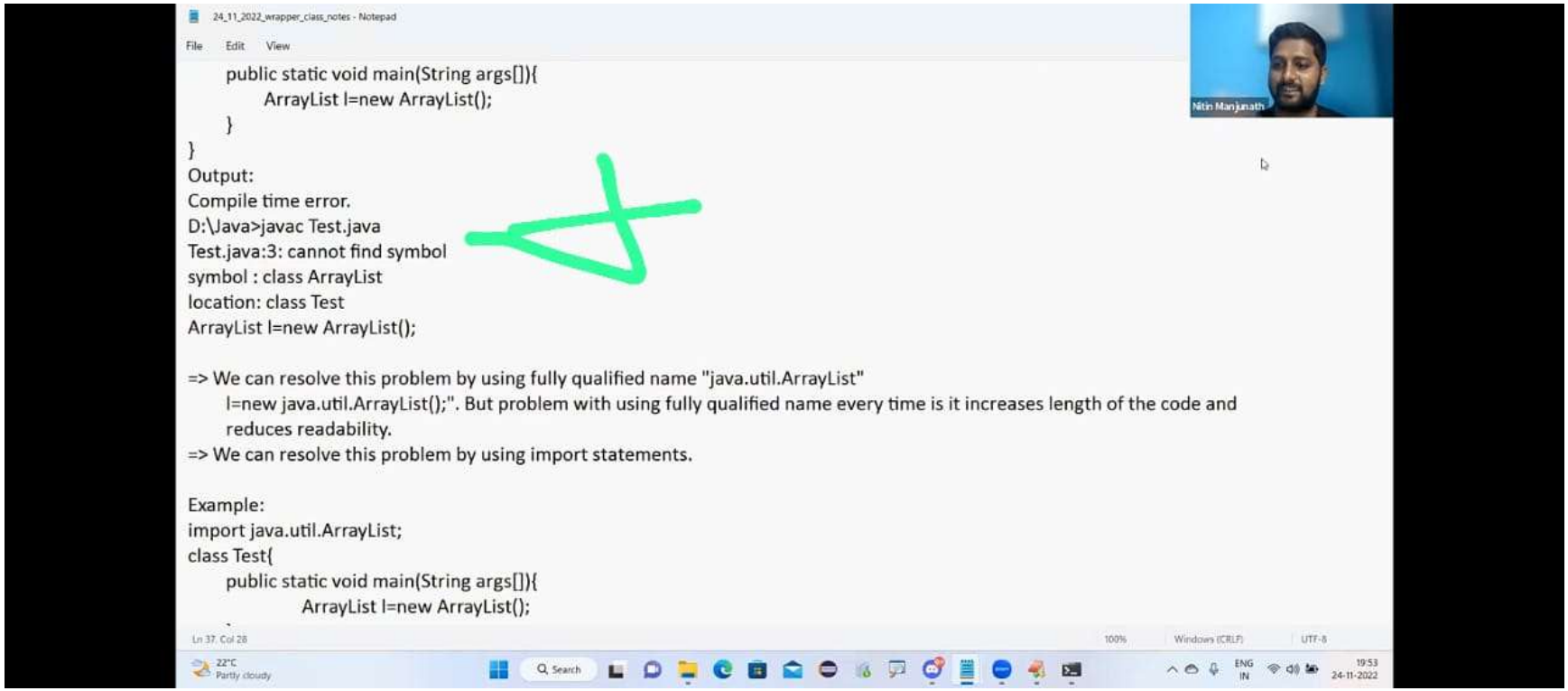
Java Import Statement Functional Interface Lamda Expression




EditPlus - [D:\Wrapper classes\Test.java]
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory Cliptext Functions
[D:] New Volume
D:\
Wrapper classes
Test.java
Java (*.java)
Test.java
For Help, press F1
Ln 20 col 82 36 00 PC ANSI
Nitin Manjunath
System.out.println("Test.class file is loading by jvm");
}
public static void main(String[] args)throws Exception{
Student student= new Student();
java.util.ArrayList al = new java.util.ArrayList();//fully qualified path
}
}
/*
javac => search for the required class information in
a. cwd
b. Did the programmer specified where the class is available
*/
22°C
Partly cloudy
Search
ENG
IN
19:47
24-11-2022


Directory Cliptext Functions
(D:) New Volume
D:\
Wrapper classes
Test.java
Java (*.java)
Test.java
For Help, press F1
Nitin Manjunath
//informing the compiler plz search for ArrayList class in "java.util"
import java.util.ArrayList;//Explicit import
import java.util.*;//implicit import
class Student
{
static{
System.out.println("Student .class file is loading");
}
Student(){
System.out.println("Student constructor is called");
}
}
public class Test
{
static{
System.out.println("Test.class file is loading by jvm");
}
22°C
Partly cloudy
Search
ENG
IN
19:52
24-11-2022


24_11_2022_wrapper_class_notes - Notepad
File Edit View
public static void main(String args[]){
ArrayList l=new ArrayList();
}
}
Output:
Compile time error.
D:\Java>javac Test.java
Test.java:3: cannot find symbol
symbol : class ArrayList
location: class Test
ArrayList l=new ArrayList();
=> We can resolve this problem by using fully qualified name "java.util.ArrayList"
l=new java.util.ArrayList();". But problem with using fully qualified name every time is it increases length of the code and
reduces readability.
=> We can resolve this problem by using import statements.
Example:
import java.util.ArrayList;
class Test{
public static void main(String args[]){
ArrayList l=new ArrayList();
Nitin Manjunath
Ln 37, Col 28
100%
Windows (CRLF)
UTF-8
22°C
Partly cloudy
Search
ENG
IN
19:53
24-11-2022

24_11_2022_wrapper_class_notes - Notepad

File Edit View

=> We can resolve this problem by using fully qualified name "java.util.ArrayList"
l=new java.util.ArrayList();". But problem with using fully qualified name every time is it increases length of the cod
reduces readability.

=> We can resolve this problem by using import statements.

Example:

```
import java.util.ArrayList;
class Test{
    public static void main(String args[]){
        ArrayList l=new ArrayList();
    }
}
```

Output:

D:\Java>javac Test.java

Hence whenever we are using import statement it is not require to use fully qualified names we can use short names directly.
This approach decreases length of the code and improves readability.

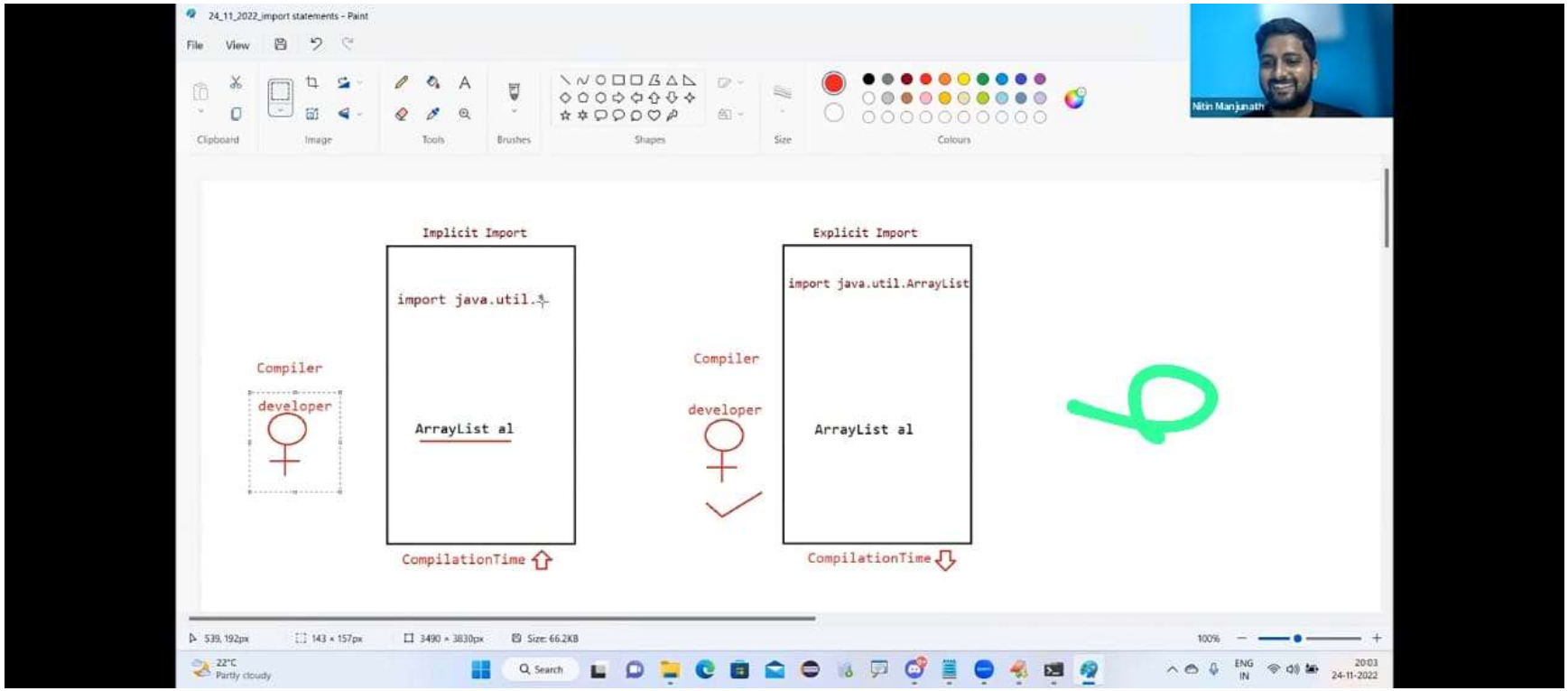
Case 1: Types of Import Statements:

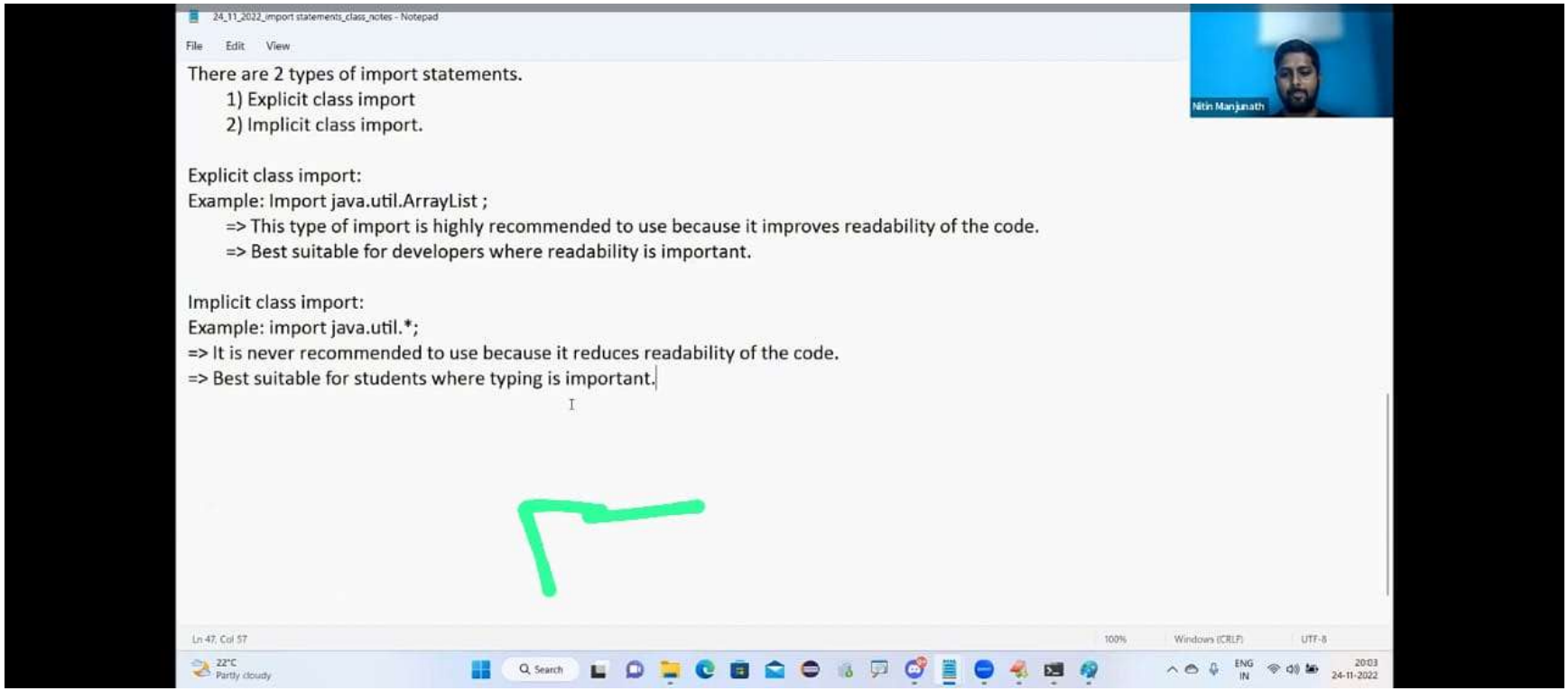
There are 2 types of import statements.

1) Explicit class import
2) Implicit class import.

Ln 27, Col 3 100% Windows (CRLF) UTF-8

22°C Partly cloudy Search ENG IN 19:54 24-11-2022


24_11_2022_import statements - Paint
File
View
Clipboard
Image
Tools
Brushes
Shapes
Size
Colours
Nitin Manjunath
Implicit Import
import java.util.*;
Compiler
developer
ArrayList al
CompilationTime
Explicit Import
import java.util.ArrayList
Compiler
developer
ArrayList al
CompilationTime
539, 192px
143 × 157px
3490 × 3830px
Size: 66.2KB
100%
22°C
Partly cloudy
Search
ENG
IN
20:03
24-11-2022


24_11_2022_import statements_class_notes - Notepad
File Edit View
There are 2 types of import statements.
1) Explicit class import
2) Implicit class import.
Explicit class import:
Example: Import java.util.ArrayList ;
=> This type of import is highly recommended to use because it improves readability of the code.
=> Best suitable for developers where readability is important.
Implicit class import:
Example: import java.util.*;
=> It is never recommended to use because it reduces readability of the code.
=> Best suitable for students where typing is important.
Nitin Manjunath
Ln 47, Col 57
100%
Windows (CRLF)
UTF-8
22°C
Partly cloudy
Search
ENG
IN
20:03
24-11-2022

24_11_2022_import statements_class_notes - Notepad

File Edit View

a. import java.util;
b. import java.util.ArrayList.*;
c. import java.util.*;
d. import java.util.ArrayList;

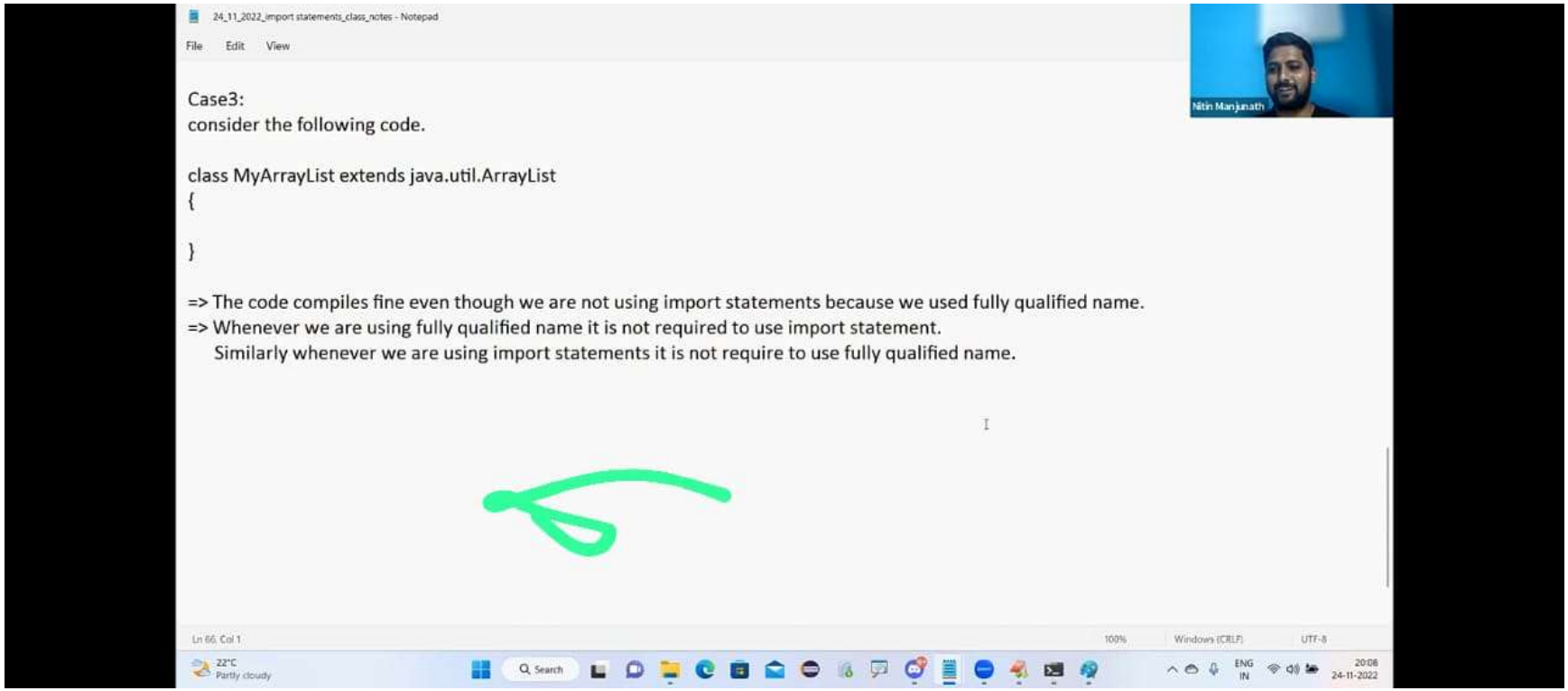
Answer: c and d

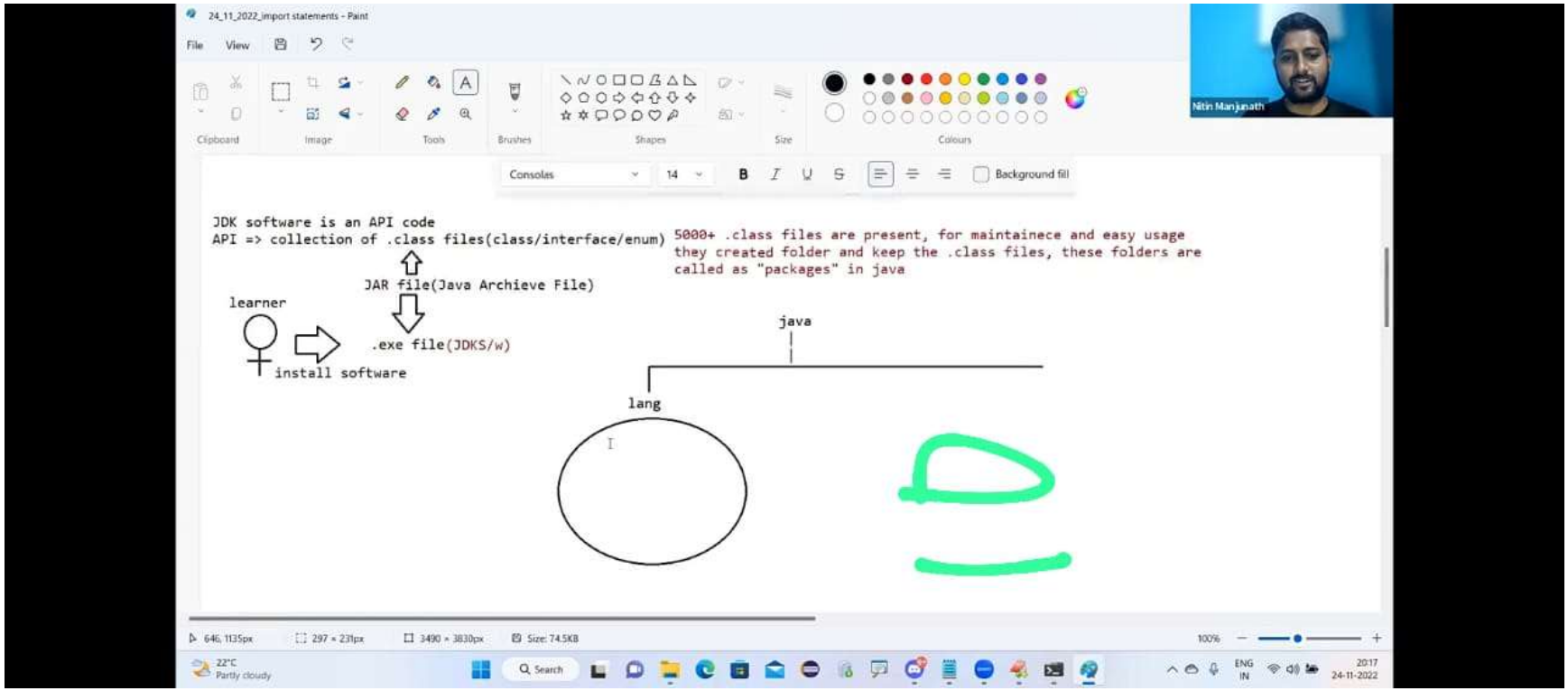
Case3:
consider the following code.

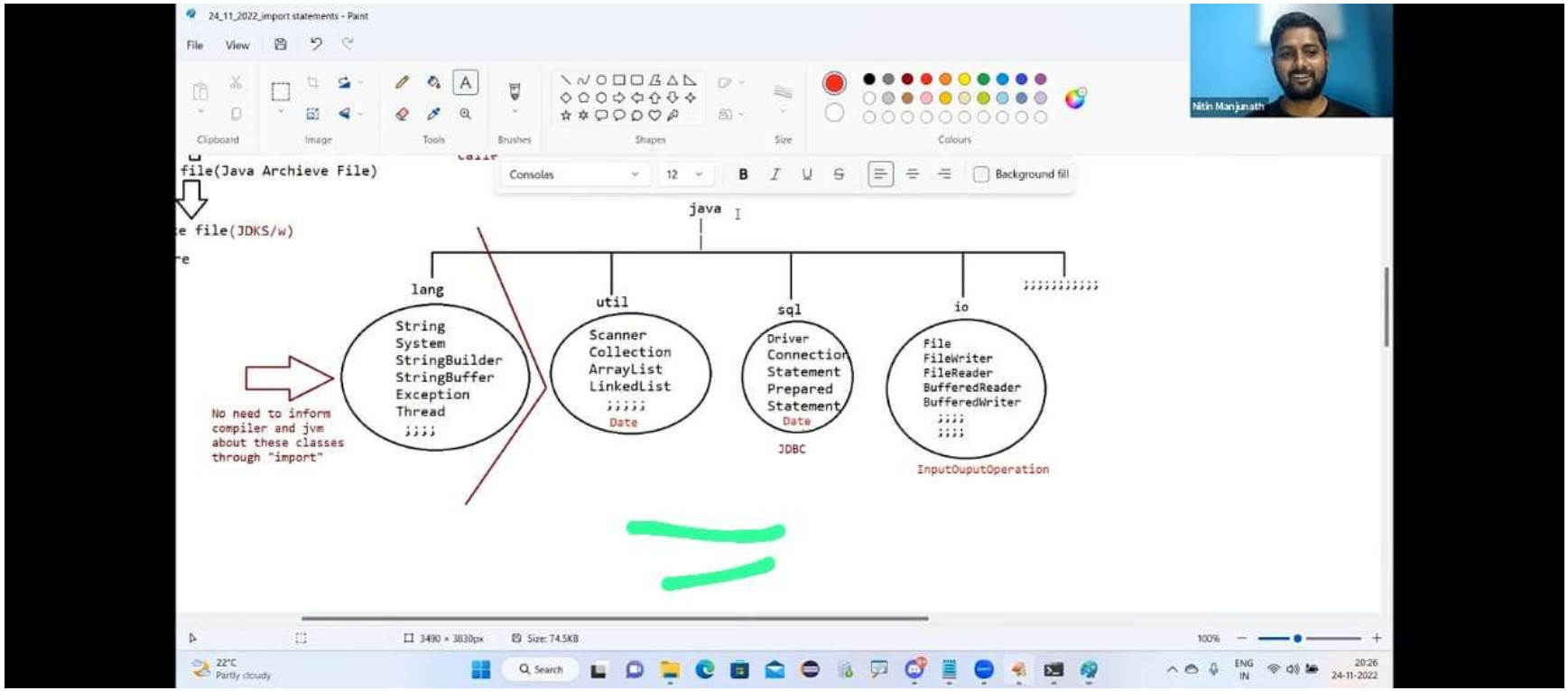
```
import java.util.ArrayList;
class MyArrayList extends java.util.ArrayList
{

}
```

Ln 62, Col 47 100% Windows (CRLF) UTF-8

22°C Partly cloudy Search ENG IN 20:07 24-11-2022


24_11_2022_import statements_class_notes - Notepad
File Edit View
Nitin Manjunath
Case3:
consider the following code.
class MyArrayList extends java.util.ArrayList
{
}
=> The code compiles fine even though we are not using import statements because we used fully qualified name.
=> Whenever we are using fully qualified name it is not required to use import statement.
Similarly whenever we are using import statements it is not require to use fully qualified name.
Ln 66, Col 1
100%
Windows (CRLF)
UTF-8
22°C
Partly cloudy
Search
ENG
IN
20:08
24-11-2022


24_11_2022_import statements - Paint
File
View
Clipboard
Image
Tools
Brushes
Shapes
Size
Colours
Nitin Manjunath
Consolas
14
Background fill
JDK software is an API code
API => collection of .class files(class/interface/enum)
5000+ .class files are present, for maintainece and easy usage
they created folder and keep the .class files, these folders are
called as "packages" in java
JAR file(Java Archieve File)
learner
.exe file(JDKS/w)
install software
java
lang
646, 1135px
297 × 231px
3490 × 3830px
Size: 74.5KB
100%
22°C
Partly cloudy
Search
ENG
IN
20:17
24-11-2022


24_11_2022_import statements - Paint
File
View
Clipboard
Image
Tools
Brushes
Shapes
Size
Colours
Nitin Manjunath
file(Java Archieve File)
e file(JDKS/w)
re
Consolas
12
Background fill
java
lang
String
System
StringBuilder
StringBuffer
Exception
Thread
;;;;
util
Scanner
Collection
ArrayList
LinkedList
;;;;;
Date
sql
Driver
Connection
Statement
Prepared
Statement
Date
JDBC
io
File
FileWriter
FileReader
BufferedReader
BufferedWriter
;;;;
;;;;
InputOuputOperation
;;;;;;;;;;;
No need to inform
compiler and jvm
about these classes
through "import"
3490 × 3830px
Size: 74.5KB
100%
22°C
Partly cloudy
Search
ENG
IN
20:26
24-11-2022

24_11_2022_import statements_class_notes - Notepad

File Edit View

Case5:

While resolving class names compiler will always gives the importance in the following order.

1. Explicit class import
2. Classes present in current working directory.
3. Implicit class import.

Example:

```
import java.util.Date;
import java.sql.*;
class Test {
    public static void main(String args[]){
        Date d=new Date();
    }
}
```

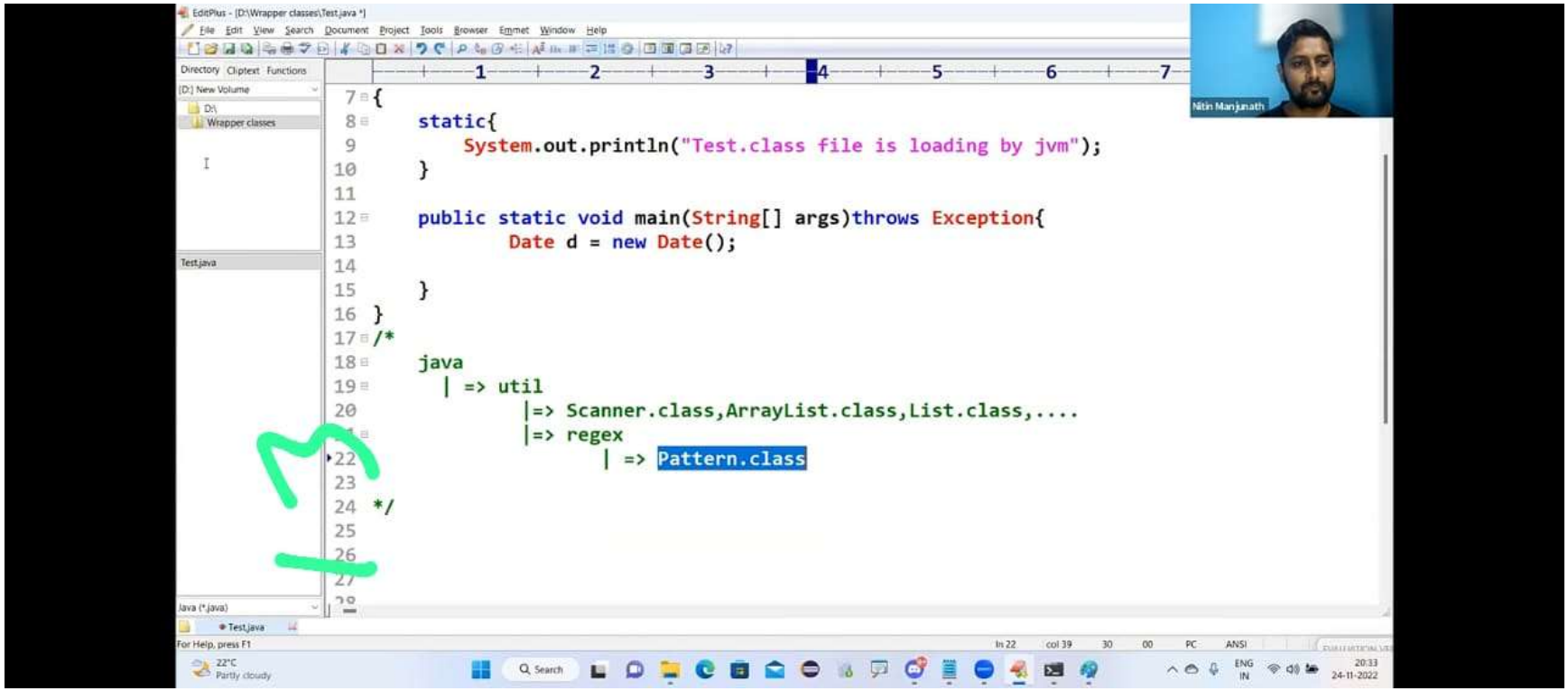
The code compiles fine and in this case util package Date will be considered.

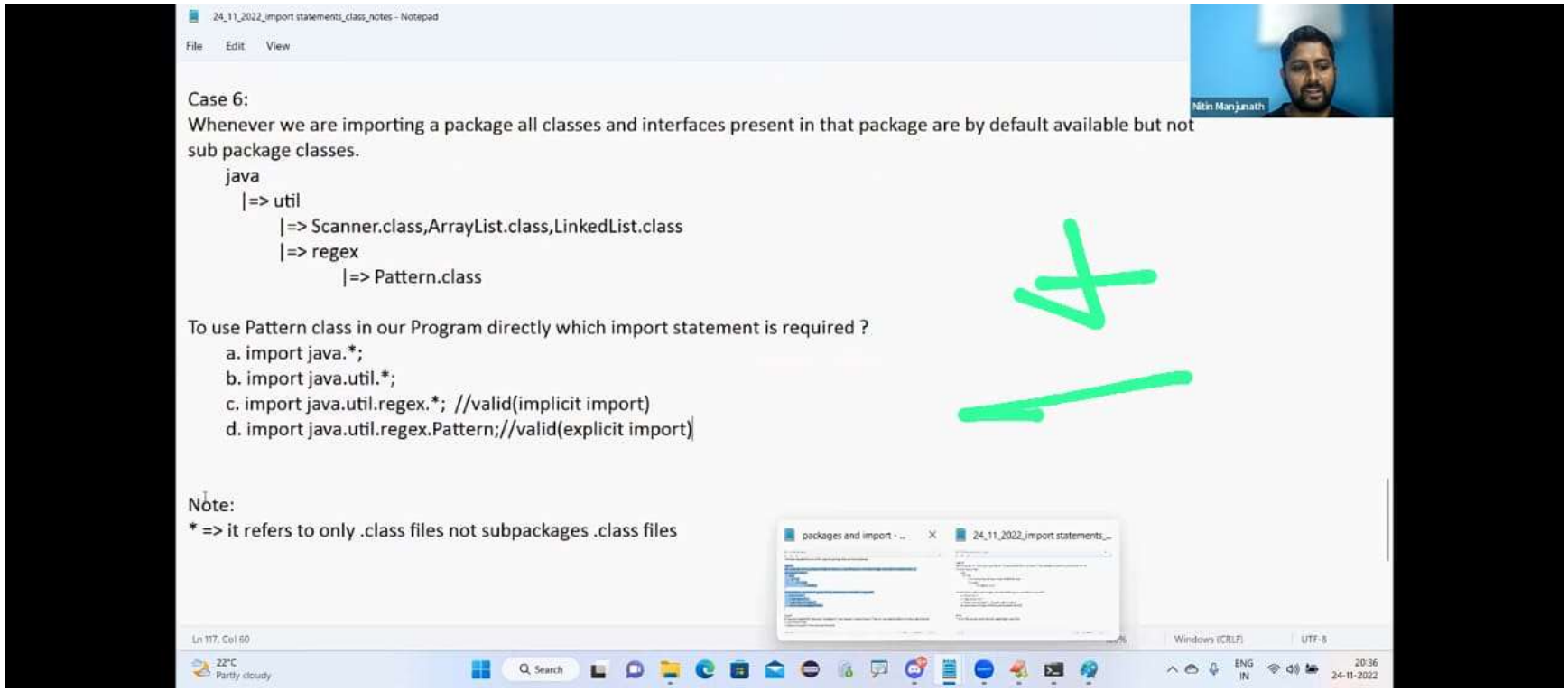
Ln 104, Col 1 100% Windows (CRLF) UTF-8

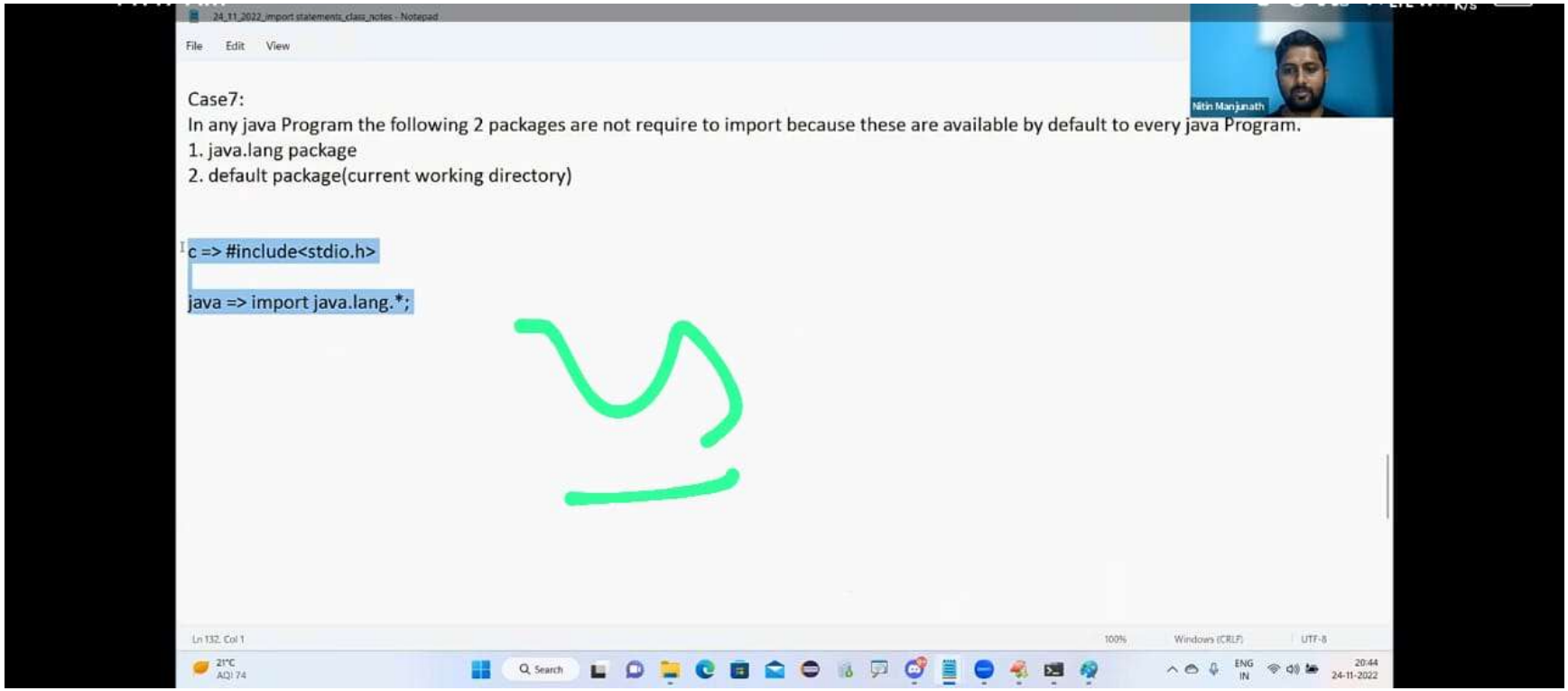
22°C Partly cloudy

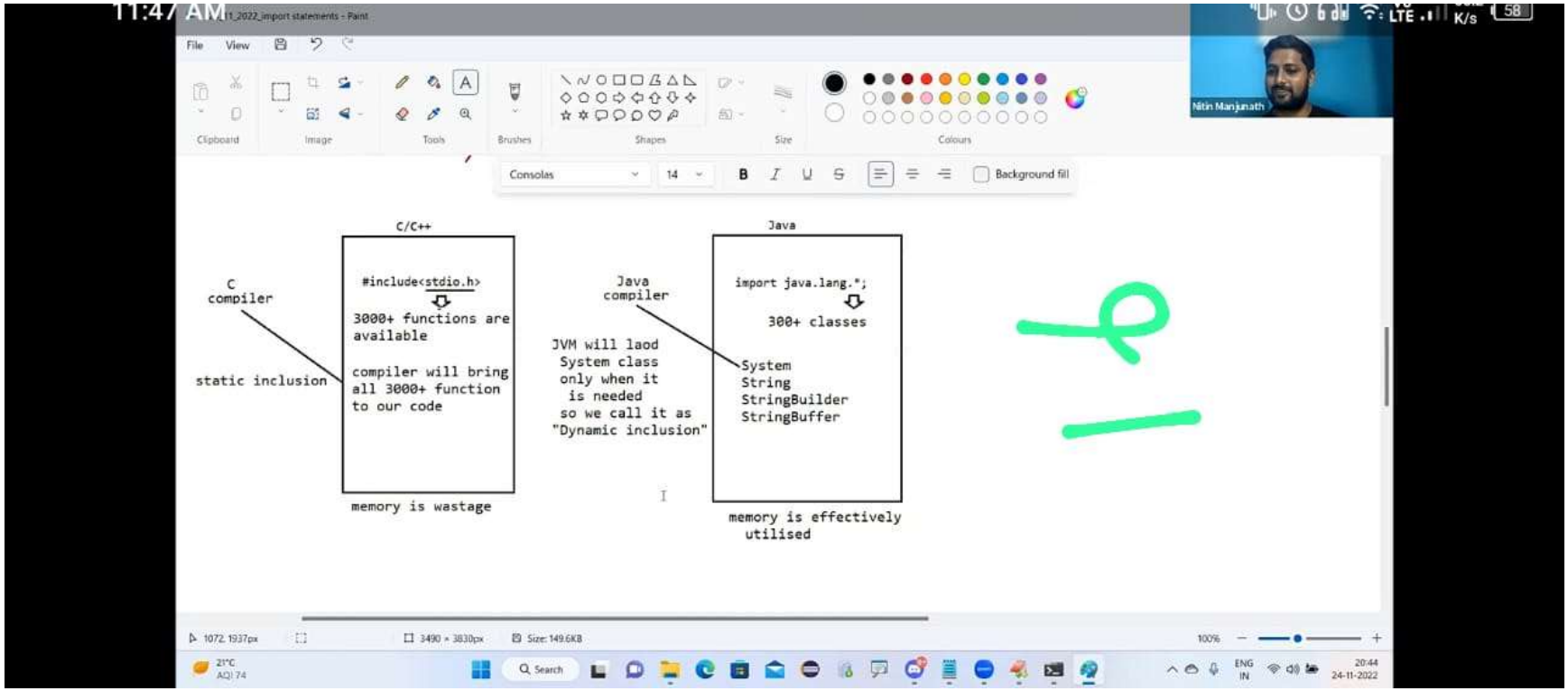
Search

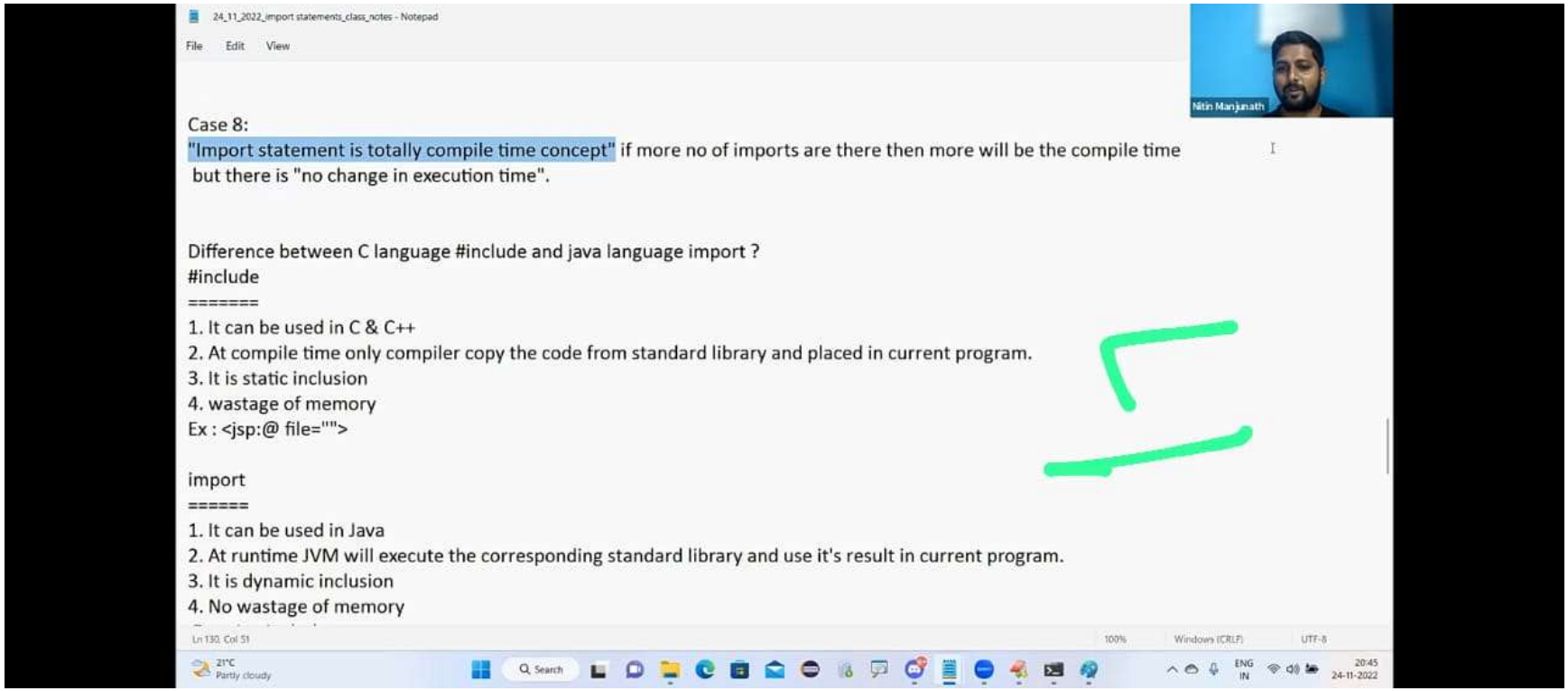
ENG IN 20:30 24-11-2022


EditPlus - [D:\Wrapper classes\Test.java *]
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory Cliptext Functions
(D:) New Volume
D:\
Wrapper classes
Test.java
Nitin Manjunath
{
static{
System.out.println("Test.class file is loading by jvm");
}
public static void main(String[] args)throws Exception{
Date d = new Date();
}
}
/*
java
| => util
|=> Scanner.class,ArrayList.class,List.class,....
|=> regex
| => Pattern.class
*/
Test.java
For Help, press F1
Ln 22 col 39 30 00 PC ANSI
22°C
Partly cloudy
Search
ENG
IN
20:33
24-11-2022


24_11_2022_import statements_class_notes - Notepad
File Edit View
Nitin Manjunath
Case 6:
Whenever we are importing a package all classes and interfaces present in that package are by default available but not sub package classes.
java
|=> util
|=> Scanner.class,ArrayList.class,LinkedList.class
|=> regex
|=> Pattern.class
To use Pattern class in our Program directly which import statement is required ?
a. import java.*;
b. import java.util.*;
c. import java.util.regex.*; //valid(implicit import)
d. import java.util.regex.Pattern;//valid(explicit import)
Note:
* => it refers to only .class files not subpackages .class files
packages and import - ...
24_11_2022_import statements_...
Ln 117, Col 60
Windows (CRLF)
UTF-8
22°C
Partly cloudy
Search
ENG
IN
20:36
24-11-2022


24_11_2022_import statements_class_notes - Notepad
File Edit View
Nitin Manjunath
Case7:
In any java Program the following 2 packages are not require to import because these are available by default to every java Program.
1. java.lang package
2. default package(current working directory)
c => #include<stdio.h>
java => import java.lang.*;
Ln 132, Col 1
100%
Windows (CRLF)
UTF-8
21°C
AQI 74
Search
ENG
IN
20:44
24-11-2022


11:47 AM
11_2022_import statements - Paint
File
View
Nitin Manjunath
Consolas
14
Background fill
C/C++
C compiler
#include<stdio.h>
3000+ functions are available
compiler will bring all 3000+ function to our code
static inclusion
memory is wastage
Java
Java compiler
import java.lang.*;
300+ classes
JVM will laod System class only when it is needed so we call it as "Dynamic inclusion"
System
String
StringBuilder
StringBuffer
memory is effectively utilised
1072, 1937px
3490 × 3830px
Size: 149.6KB
100%
Search
20:44
24-11-2022


24_11_2022_import statements_class_notes - Notepad
File Edit View
Nitin Manjunath
Case 8:
"Import statement is totally compile time concept" if more no of imports are there then more will be the compile time but there is "no change in execution time".
Difference between C language #include and java language import ?
#include
=======
1. It can be used in C & C++
2. At compile time only compiler copy the code from standard library and placed in current program.
3. It is static inclusion
4. wastage of memory
Ex : <jsp:@ file="">
import
======
1. It can be used in Java
2. At runtime JVM will execute the corresponding standard library and use it's result in current program.
3. It is dynamic inclusion
4. No wastage of memory
Ln 130, Col 51
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
20:45
24-11-2022

24_11_2022_import statements_class_notes - Notepad

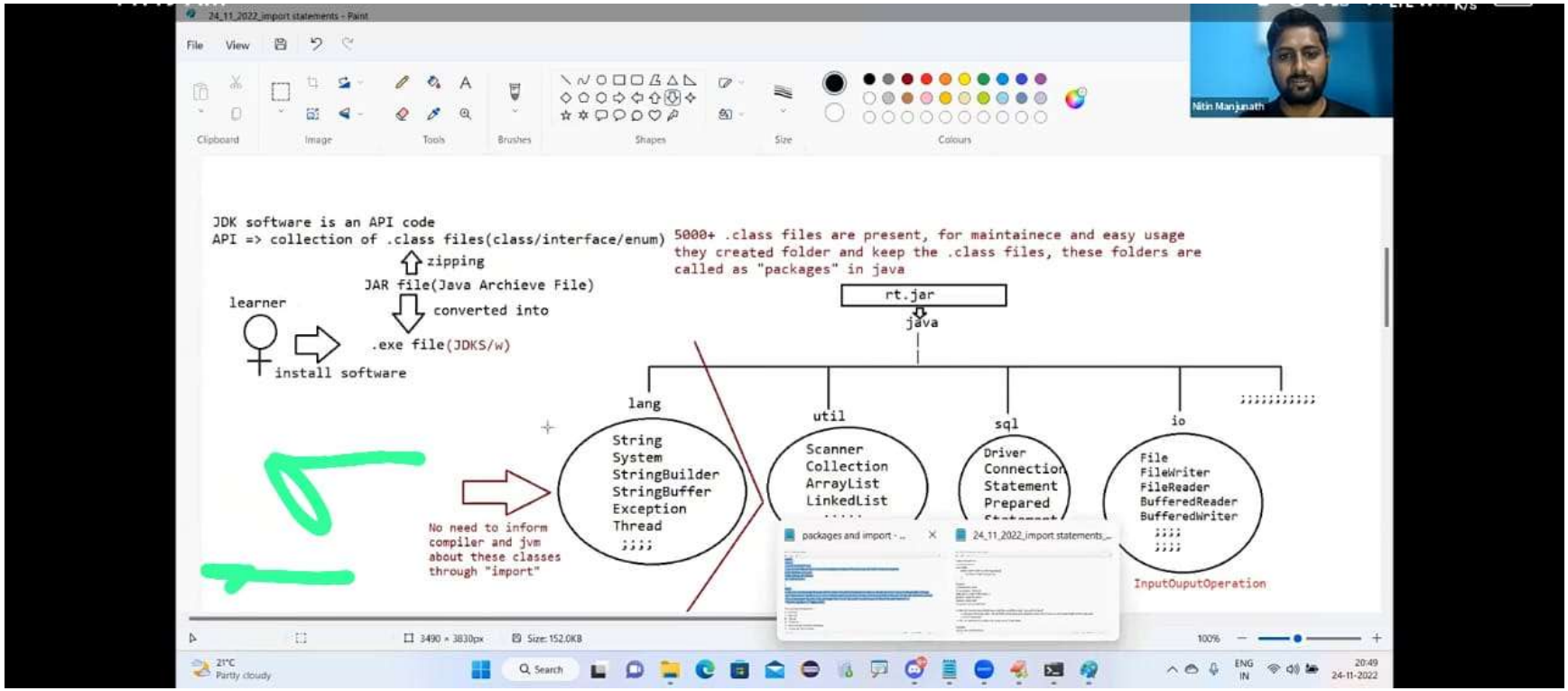
File Edit View

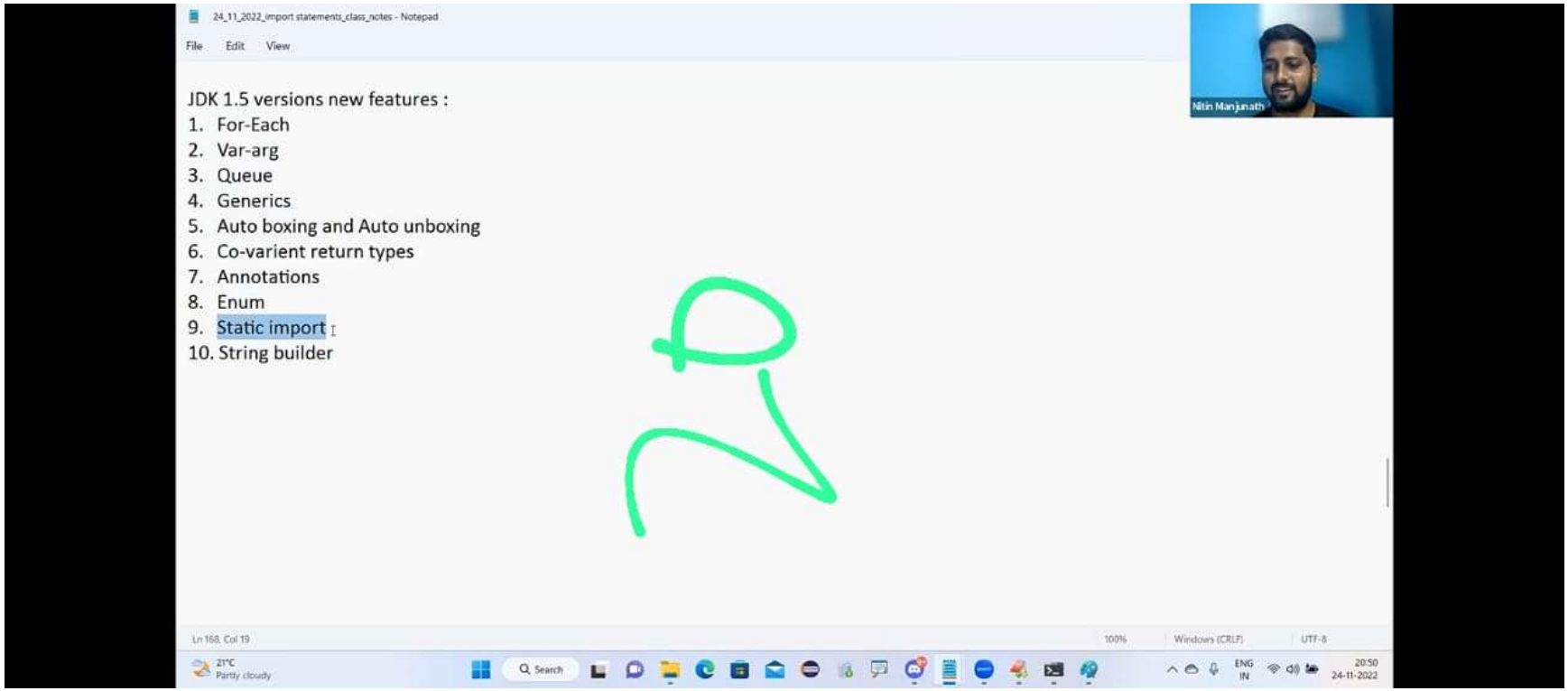
4. No wastage of memory
Ex : <jsp:include >

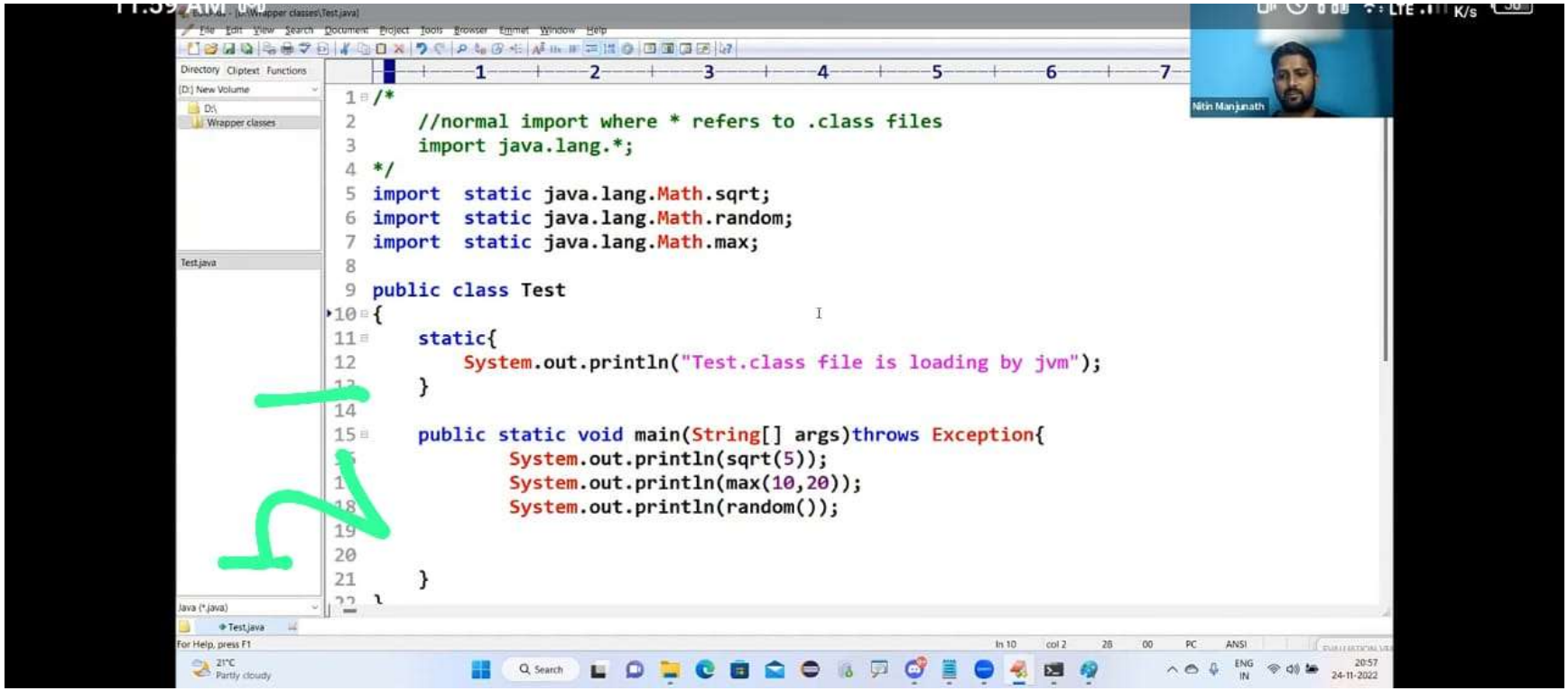
Note:
In the case of C language #include all the header files will be loaded at the time of include statement hence it follows static loading.
But in java import statement no ".class" will be loaded at the time of import statements in the next lines of the code whenever we are using a particular class then only corresponding ".class" file will be loaded. Hence it follows "dynamic loading" or "load-on -demand" or "load-on-fly".

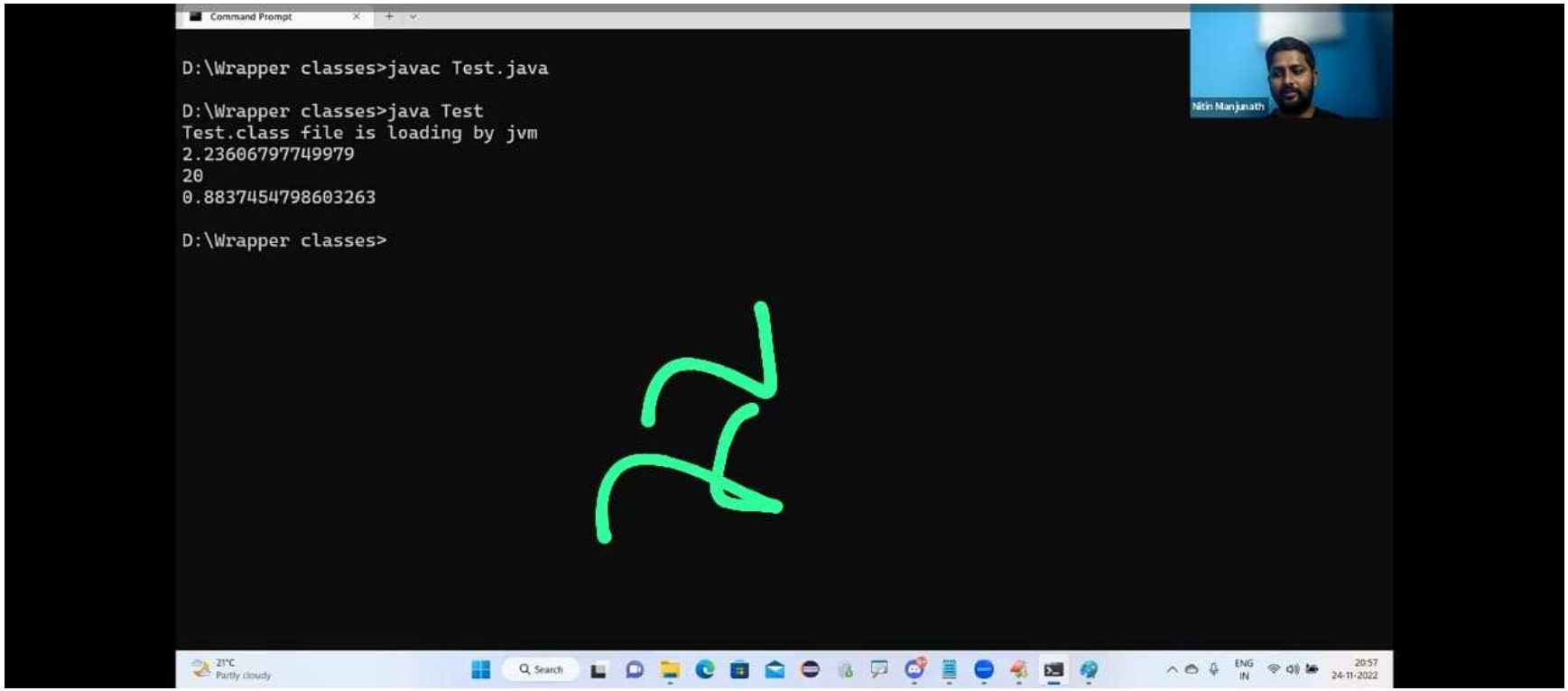
Ln 157, Col 1 100% Windows (CRLF) UTF-8

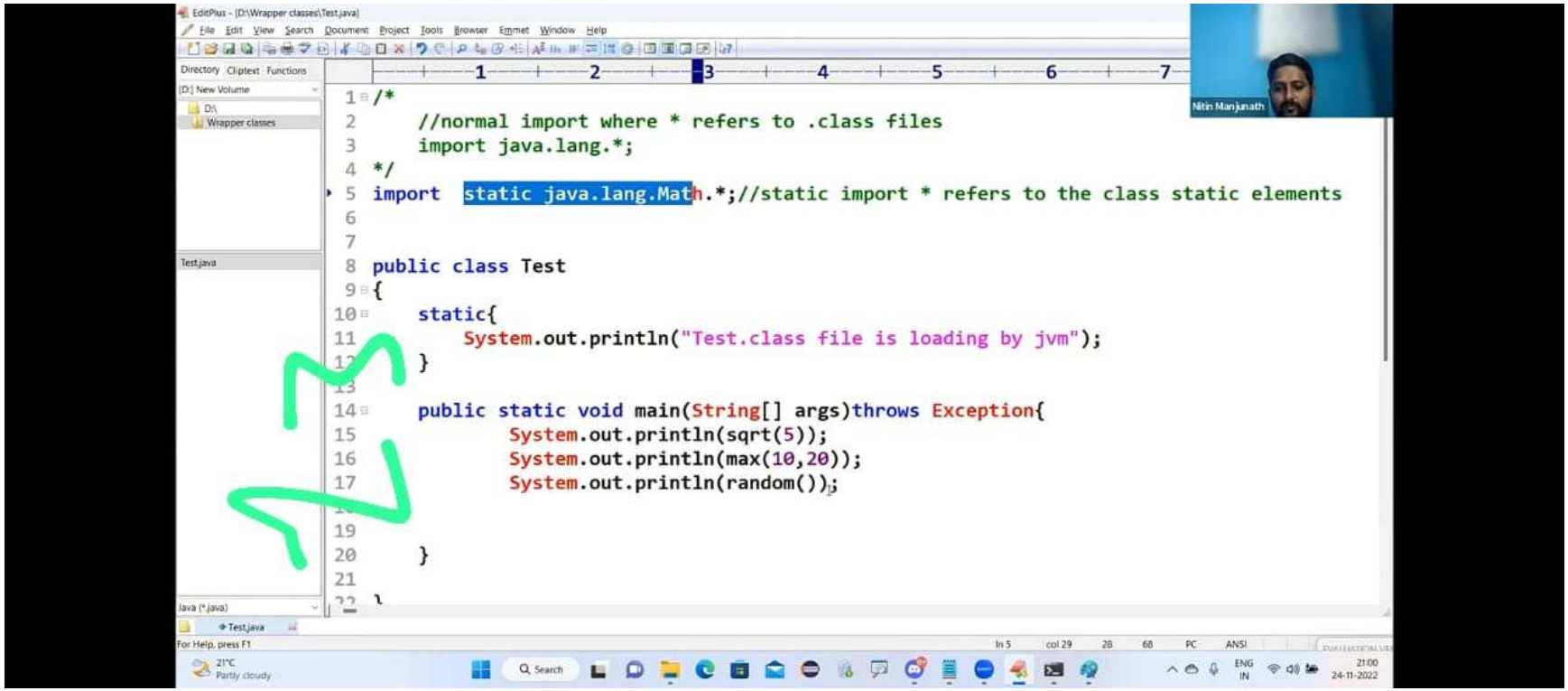
21°C Partly cloudy Search ENG IN 20:46 24-11-2022

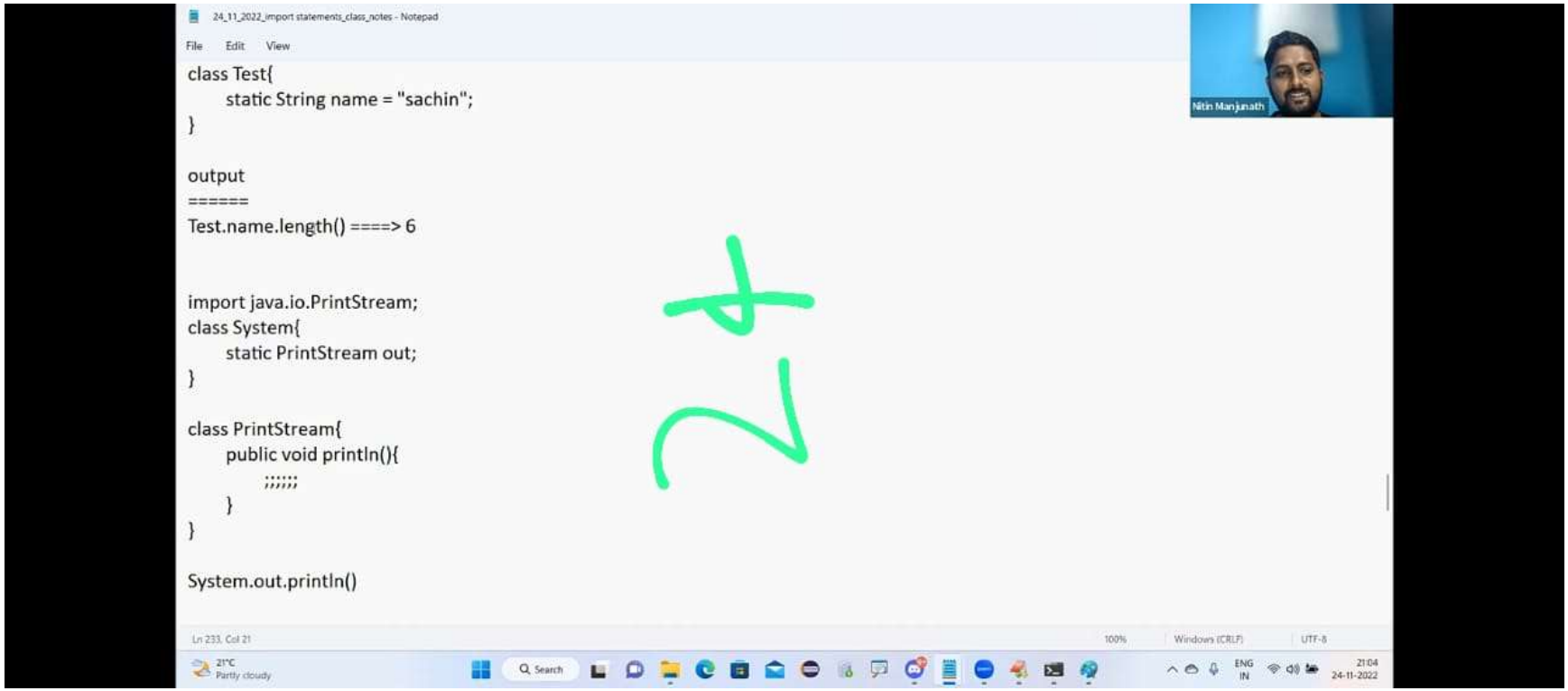

24_11_2022_import statements - Paint
File
View
Clipboard
Image
Tools
Brushes
Shapes
Size
Colours
Nitin Manjunath
JDK software is an API code
API => collection of .class files(class/interface/enum)
zipping
JAR file(Java Archieve File)
converted into
.exe file(JDKS/w)
learner
install software
5000+ .class files are present, for maintainece and easy usage
they created folder and keep the .class files, these folders are
called as "packages" in java
rt.jar
java
lang
String
System
StringBuilder
StringBuffer
Exception
Thread
;;;;
util
Scanner
Collection
ArrayList
LinkedList
sql
Driver
Connection
Statement
Prepared
io
File
FileWriter
FileReader
BufferedReader
BufferedWriter
;;;;
;;;;
;;;;;;;;;;;;
No need to inform
compiler and jvm
about these classes
through "import"
InputOuputOperation
packages and import - ...
24_11_2022_import statements_...
3490 × 3830px
Size: 152.0KB
100%
21°C
Partly cloudy
Search
ENG
IN
20:49
24-11-2022

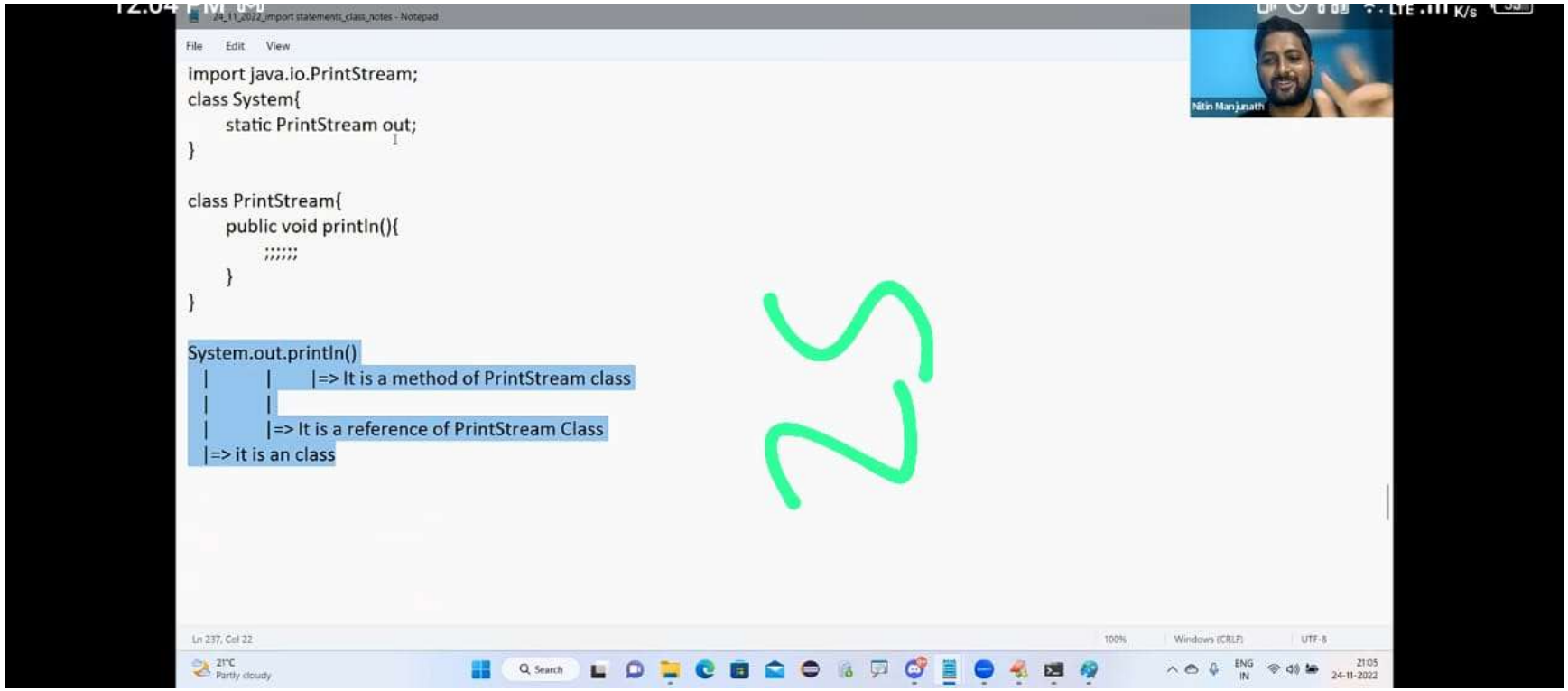

24_11_2022_import statements_class_notes - Notepad
File Edit View
JDK 1.5 versions new features :
1. For-Each
2. Var-arg
3. Queue
4. Generics
5. Auto boxing and Auto unboxing
6. Co-varient return types
7. Annotations
8. Enum
9. Static import
10. String builder
Nitin Manjunath
Ln 168, Col 19
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
20:50
24-11-2022

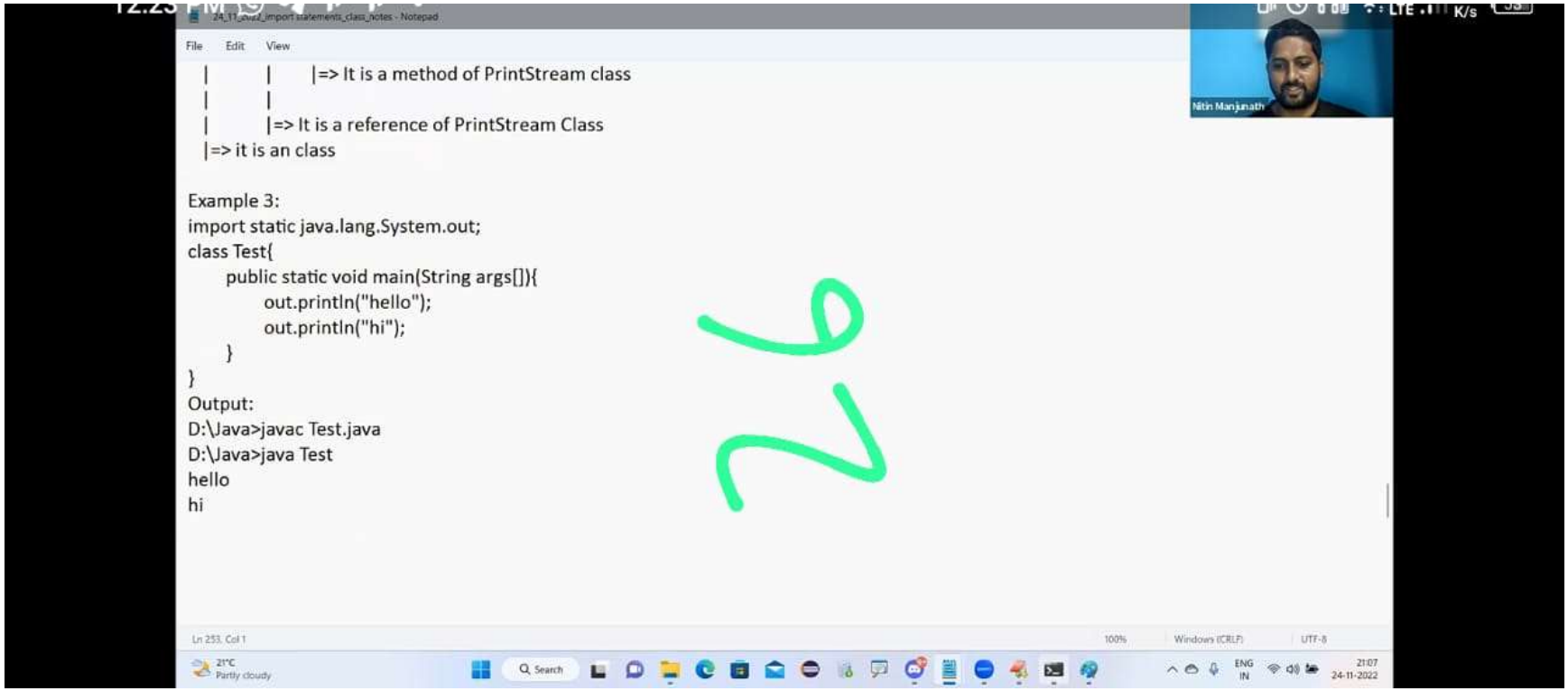

/*
//normal import where * refers to .class files
import java.lang.*;
*/
import static java.lang.Math.sqrt;
import static java.lang.Math.random;
import static java.lang.Math.max;
public class Test
{
static{
System.out.println("Test.class file is loading by jvm");
}
public static void main(String[] args)throws Exception{
System.out.println(sqrt(5));
System.out.println(max(10,20));
System.out.println(random());
}
Nitin Manjunath

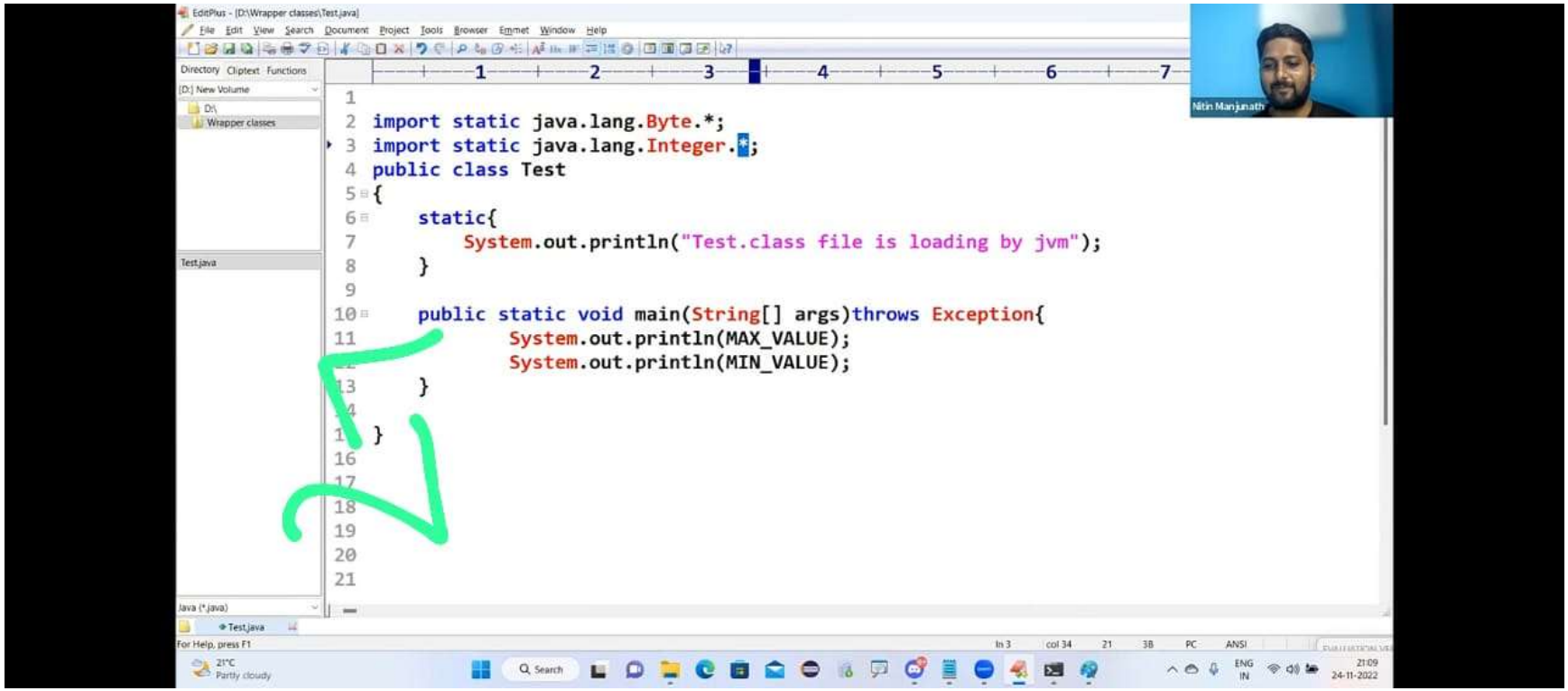

Command Prompt
D:\Wrapper classes>javac Test.java
D:\Wrapper classes>java Test
Test.class file is loading by jvm
2.23606797749979
20
0.8837454798603263
D:\Wrapper classes>
Nitin Manjunath
21°C
Partly cloudy
Search
ENG
IN
20:57
24-11-2022

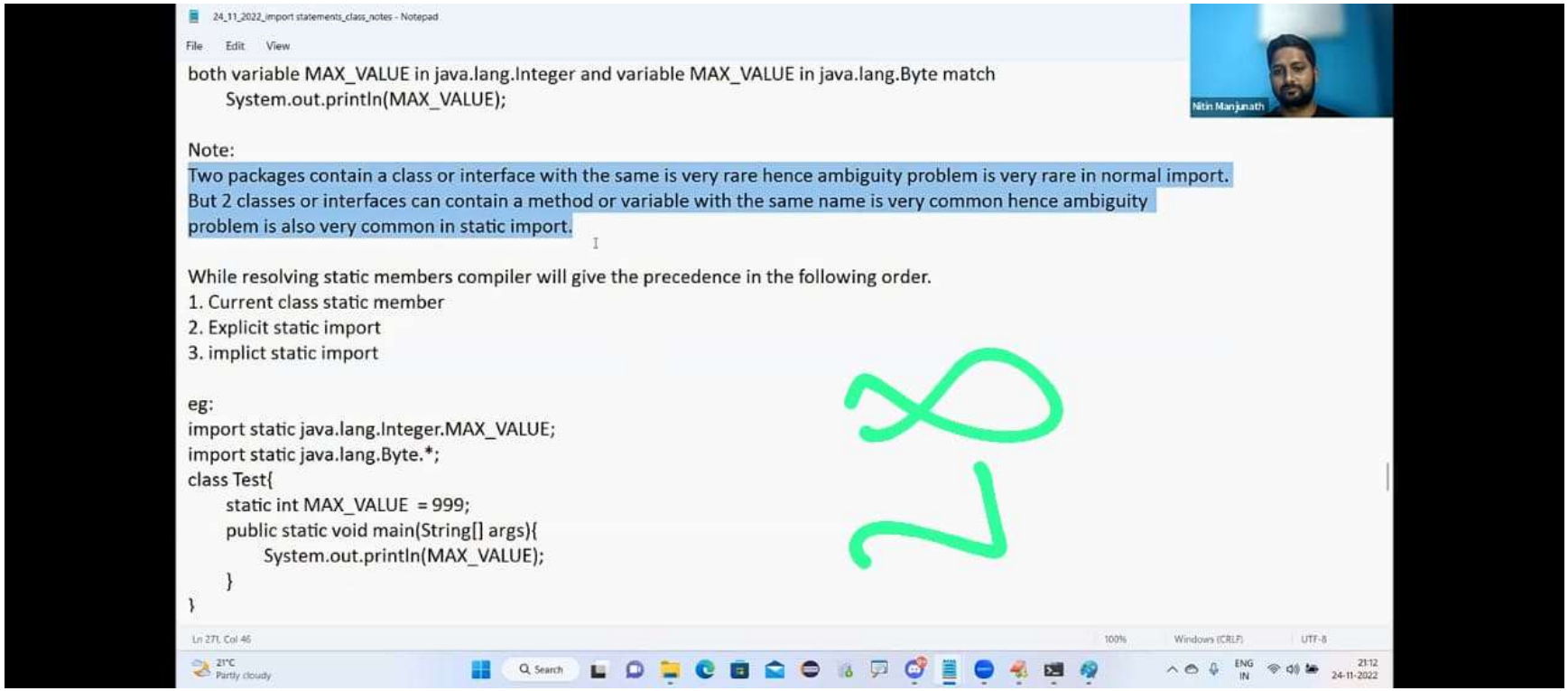

EditPlus - [D:\Wrapper classes\Test.java]
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory Cliptext Functions
[D:] New Volume
D:\
Wrapper classes
Test.java
Java (*.java)
Test.java
For Help, press F1
Nitin Manjunath
/*
//normal import where * refers to .class files
import java.lang.*;
*/
import static java.lang.Math.*;//static import * refers to the class static elements
public class Test
{
static{
System.out.println("Test.class file is loading by jvm");
}
public static void main(String[] args)throws Exception{
System.out.println(sqrt(5));
System.out.println(max(10,20));
System.out.println(random());
}
}
Ln 5 col 29 28 68 PC ANSI
21°C
Partly cloudy
Search
ENG
IN
21:00
24-11-2022

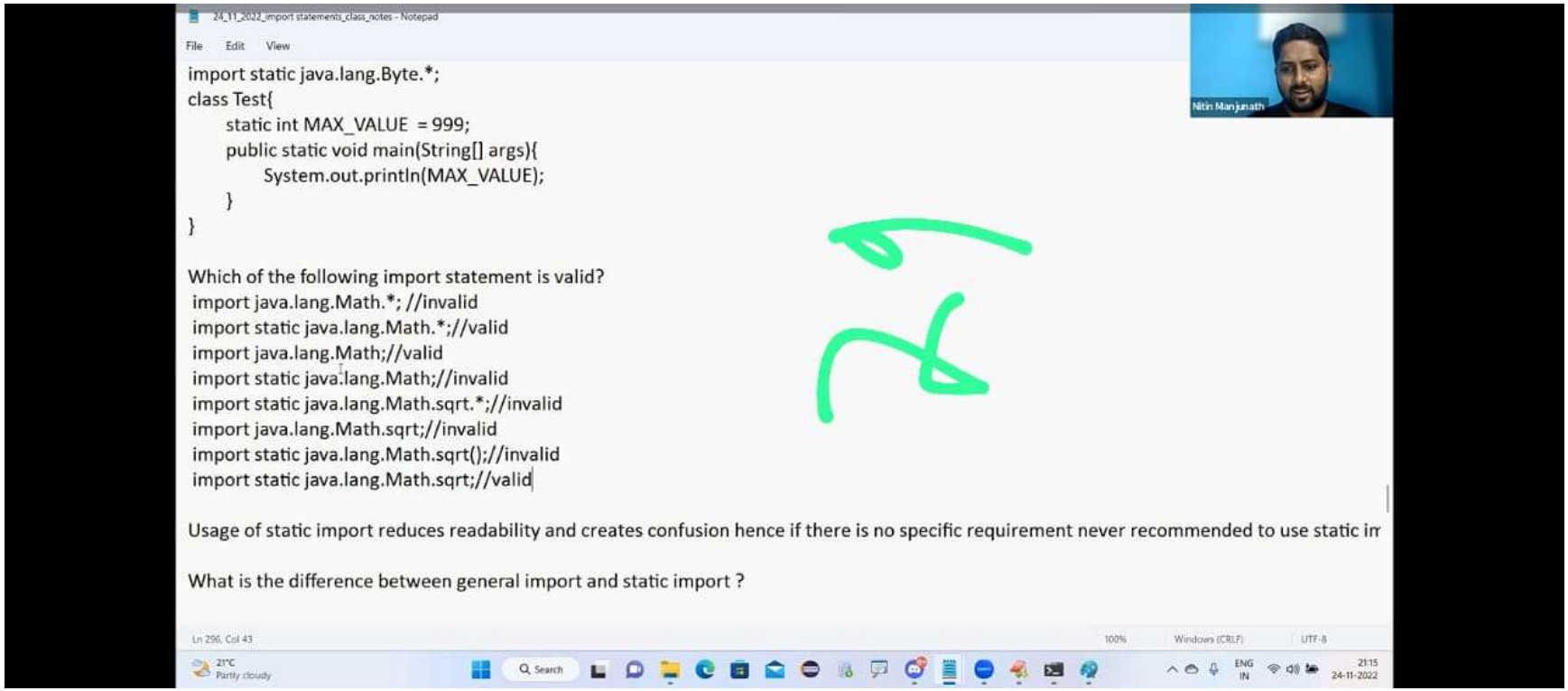

24_11_2022_import statements_class_notes - Notepad
File Edit View
class Test{
static String name = "sachin";
}
output
======
Test.name.length() ====> 6
import java.io.PrintStream;
class System{
static PrintStream out;
}
class PrintStream{
public void println(){
;;;;;;
}
}
System.out.println()
Nitin Manjunath
24
Ln 233, Col 21
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
21:04
24-11-2022

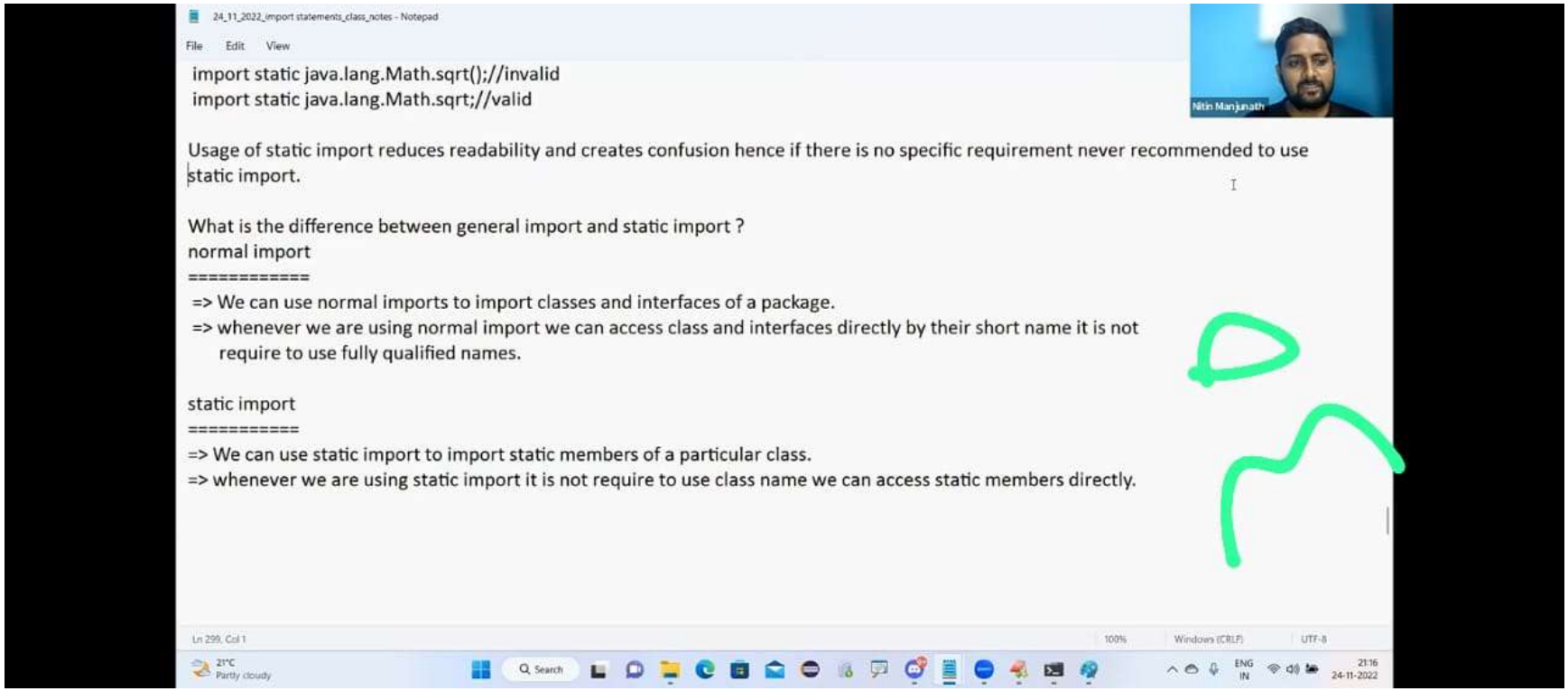

24_11_2022_import statements_class_notes - Notepad
File Edit View
import java.io.PrintStream;
class System{
static PrintStream out;
}
class PrintStream{
public void println(){
;;;;;;
}
}
System.out.println()
| | |=> It is a method of PrintStream class
| |
| |=> It is a reference of PrintStream Class
|=> it is an class
Nitin Manjunath
Ln 237, Col 22
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
21:05
24-11-2022

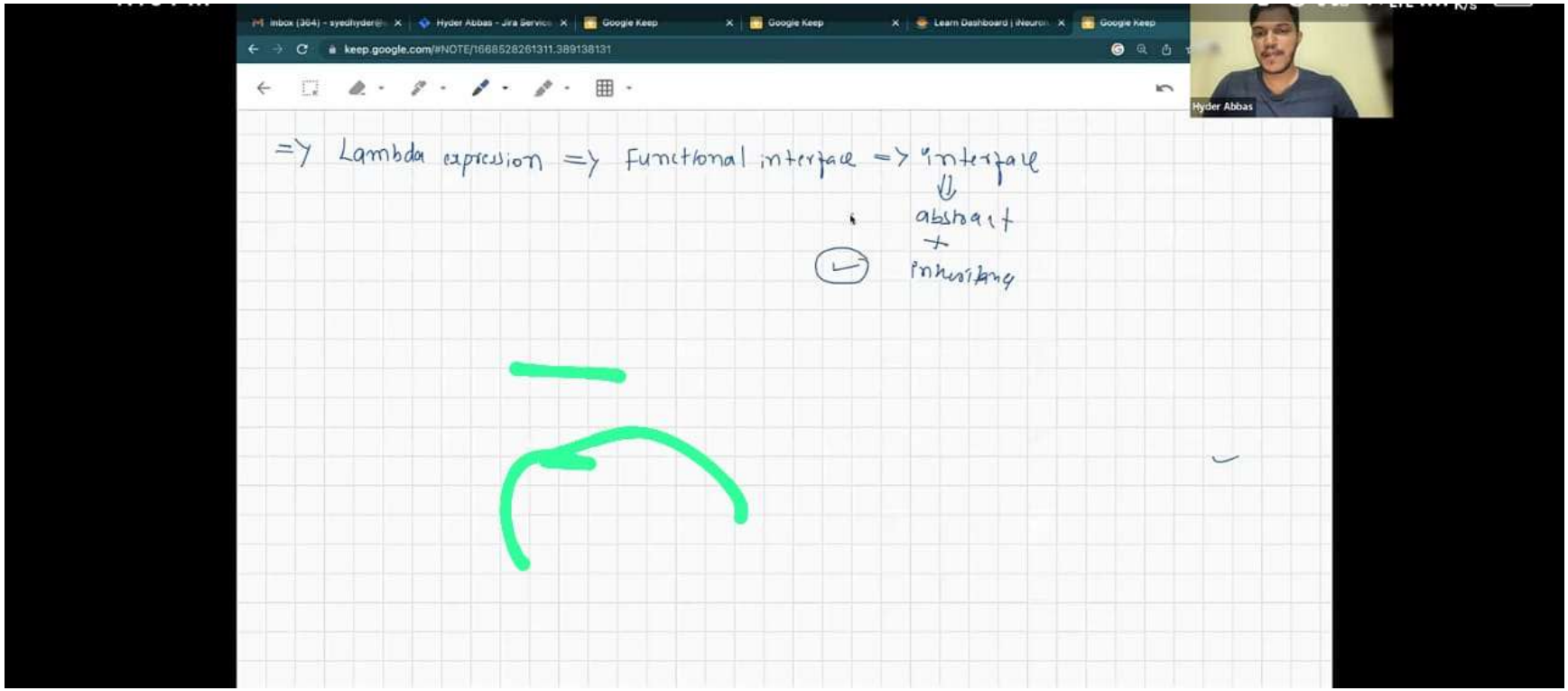

24_11_java2_import statements_class_notes - Notepad
File Edit View
| | |=> It is a method of PrintStream class
| |
| |=> It is a reference of PrintStream Class
|=> it is an class
Example 3:
import static java.lang.System.out;
class Test{
public static void main(String args[]){
out.println("hello");
out.println("hi");
}
}
Output:
D:\Java>javac Test.java
D:\Java>java Test
hello
hi
Nitin Manjunath
Ln 253, Col 1
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
21:07
24-11-2022

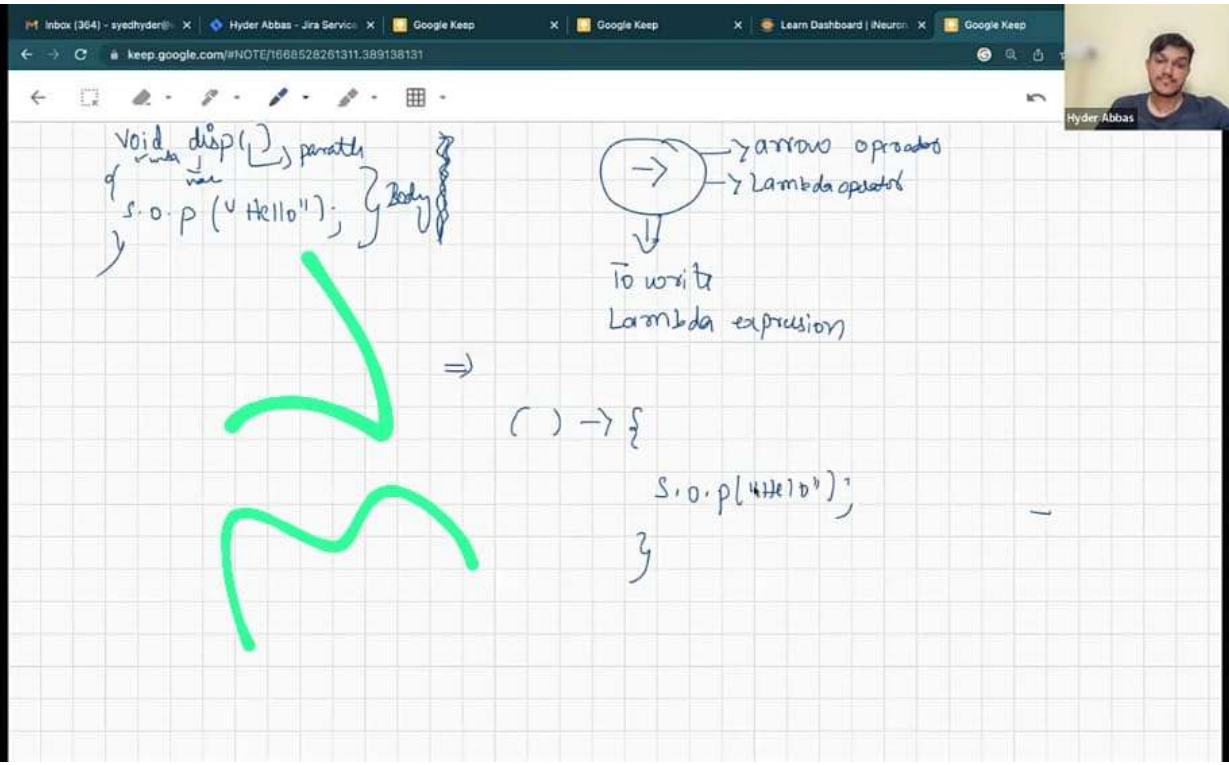

EditPlus - [D:\Wrapper classes\Test.java]
File Edit View Search Document Project Tools Browser Emmet Window Help
Directory Cliptext Functions
[D:] New Volume
D:\
Wrapper classes
Test.java
Java (*.java)
Test.java
For Help, press F1
Ln 3 col 34 21 3B PC ANSI
Nitin Manjunath
1 2 3 4 5 6 7
import static java.lang.Byte.*;
import static java.lang.Integer.*;
public class Test
{
static{
System.out.println("Test.class file is loading by jvm");
}
public static void main(String[] args)throws Exception{
System.out.println(MAX_VALUE);
System.out.println(MIN_VALUE);
}
}
21°C
Partly cloudy
Search
ENG
IN
21:09
24-11-2022

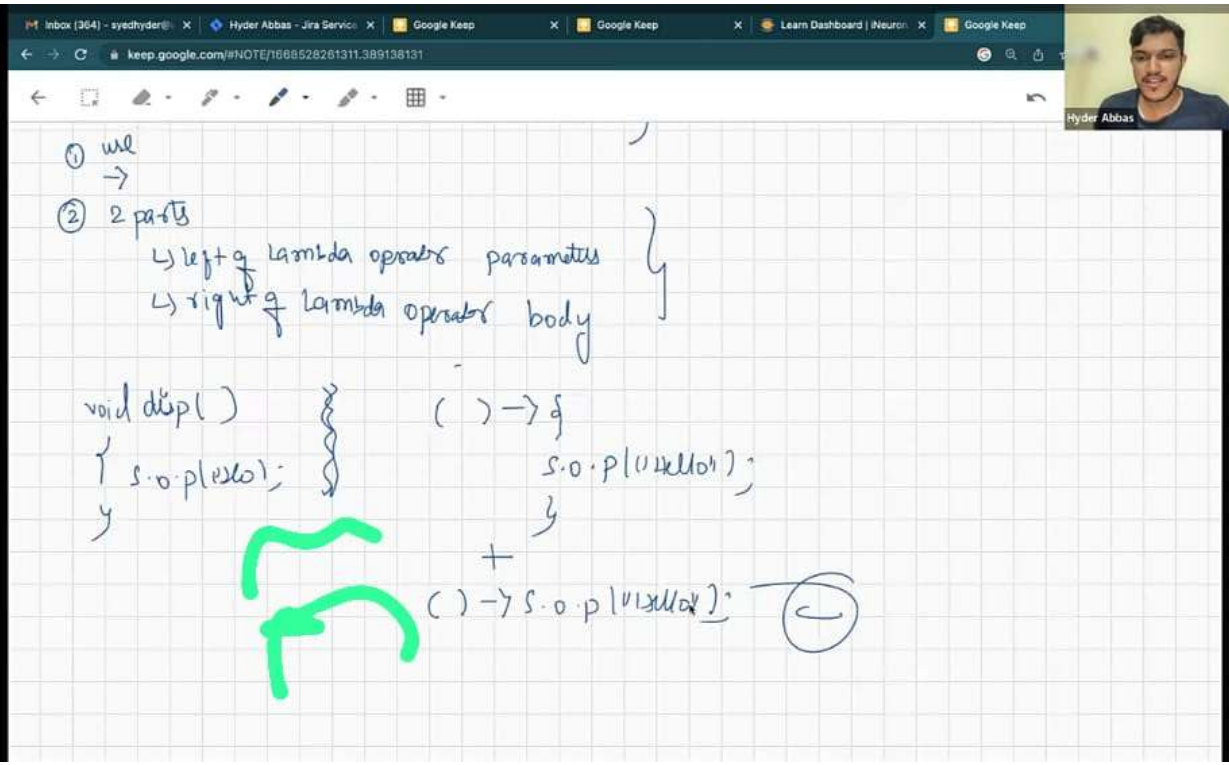

24_11_2022_import statements_class_notes - Notepad
File Edit View
both variable MAX_VALUE in java.lang.Integer and variable MAX_VALUE in java.lang.Byte match
System.out.println(MAX_VALUE);
Note:
Two packages contain a class or interface with the same is very rare hence ambiguity problem is very rare in normal import.
But 2 classes or interfaces can contain a method or variable with the same name is very common hence ambiguity problem is also very common in static import.
While resolving static members compiler will give the precedence in the following order.
1. Current class static member
2. Explicit static import
3. implict static import
eg:
import static java.lang.Integer.MAX_VALUE;
import static java.lang.Byte.*;
class Test{
static int MAX_VALUE = 999;
public static void main(String[] args){
System.out.println(MAX_VALUE);
}
}
Nitin Manjunath
Ln 271, Col 46
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
21:12
24-11-2022

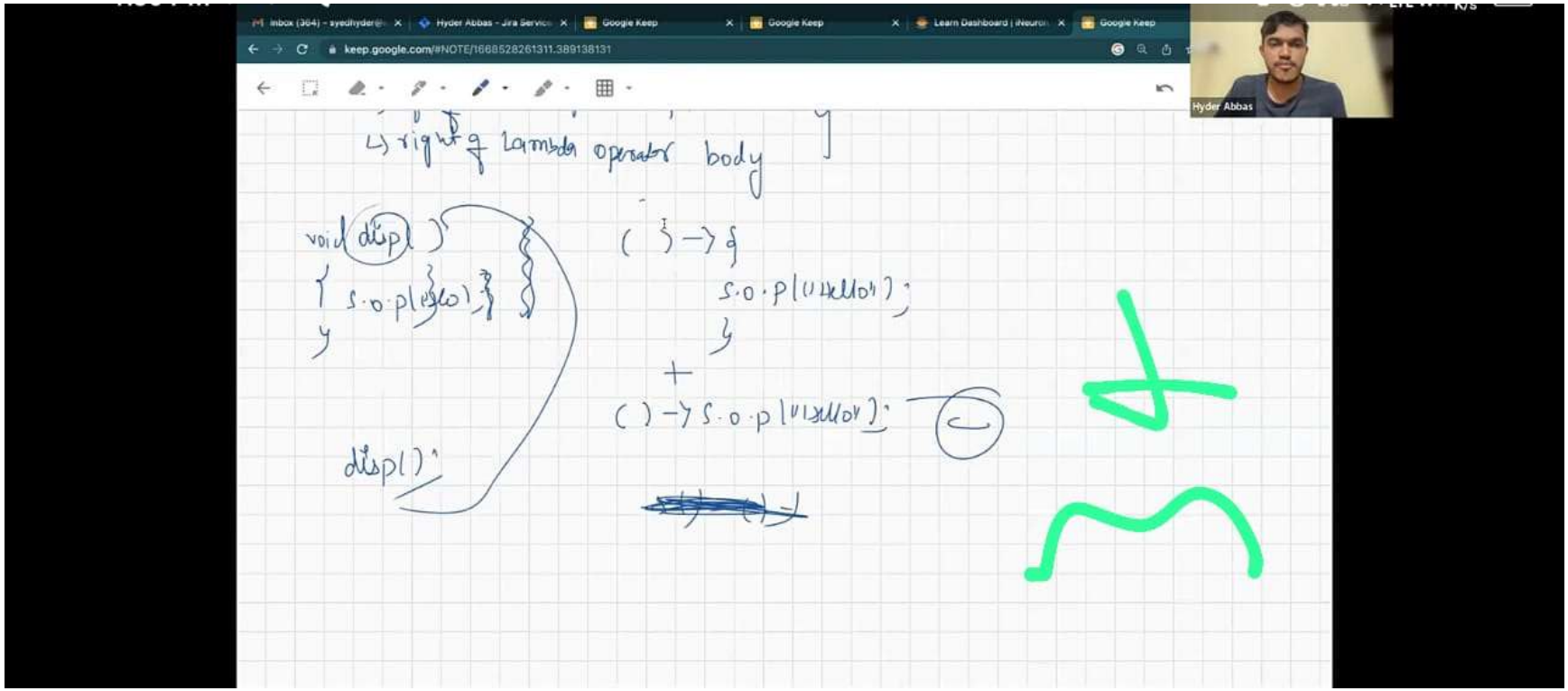

24_11_2022_import statements_class_notes - Notepad
File Edit View
import static java.lang.Byte.*;
class Test{
static int MAX_VALUE = 999;
public static void main(String[] args){
System.out.println(MAX_VALUE);
}
}
Which of the following import statement is valid?
import java.lang.Math.*; //invalid
import static java.lang.Math.*;//valid
import java.lang.Math;//valid
import static java.lang.Math;//invalid
import static java.lang.Math.sqrt.*;//invalid
import java.lang.Math.sqrt;//invalid
import static java.lang.Math.sqrt();//invalid
import static java.lang.Math.sqrt;//valid
Usage of static import reduces readability and creates confusion hence if there is no specific requirement never recommended to use static im
What is the difference between general import and static import ?
Nitin Manjunath
Ln 296, Col 43
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
21:15
24-11-2022

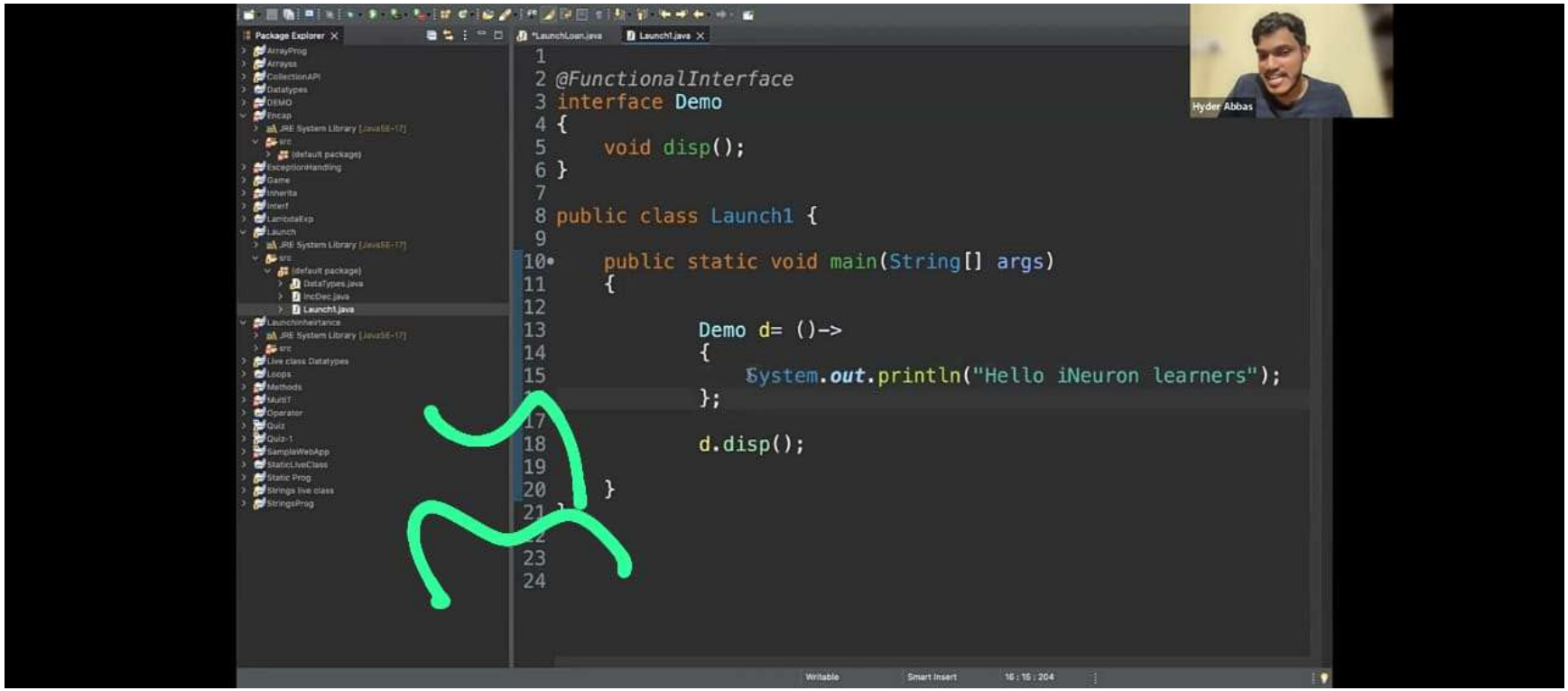

24_11_2022_import statements_class_notes - Notepad
File Edit View
import static java.lang.Math.sqrt();//invalid
import static java.lang.Math.sqrt;//valid
Usage of static import reduces readability and creates confusion hence if there is no specific requirement never recommended to use static import.
What is the difference between general import and static import ?
normal import
=> We can use normal imports to import classes and interfaces of a package.
=> whenever we are using normal import we can access class and interfaces directly by their short name it is not require to use fully qualified names.
static import
=> We can use static import to import static members of a particular class.
=> whenever we are using static import it is not require to use class name we can access static members directly.
Nitin Manjunath
Ln 299, Col 1
100%
Windows (CRLF)
UTF-8
21°C
Partly cloudy
Search
ENG
IN
21:16
24-11-2022

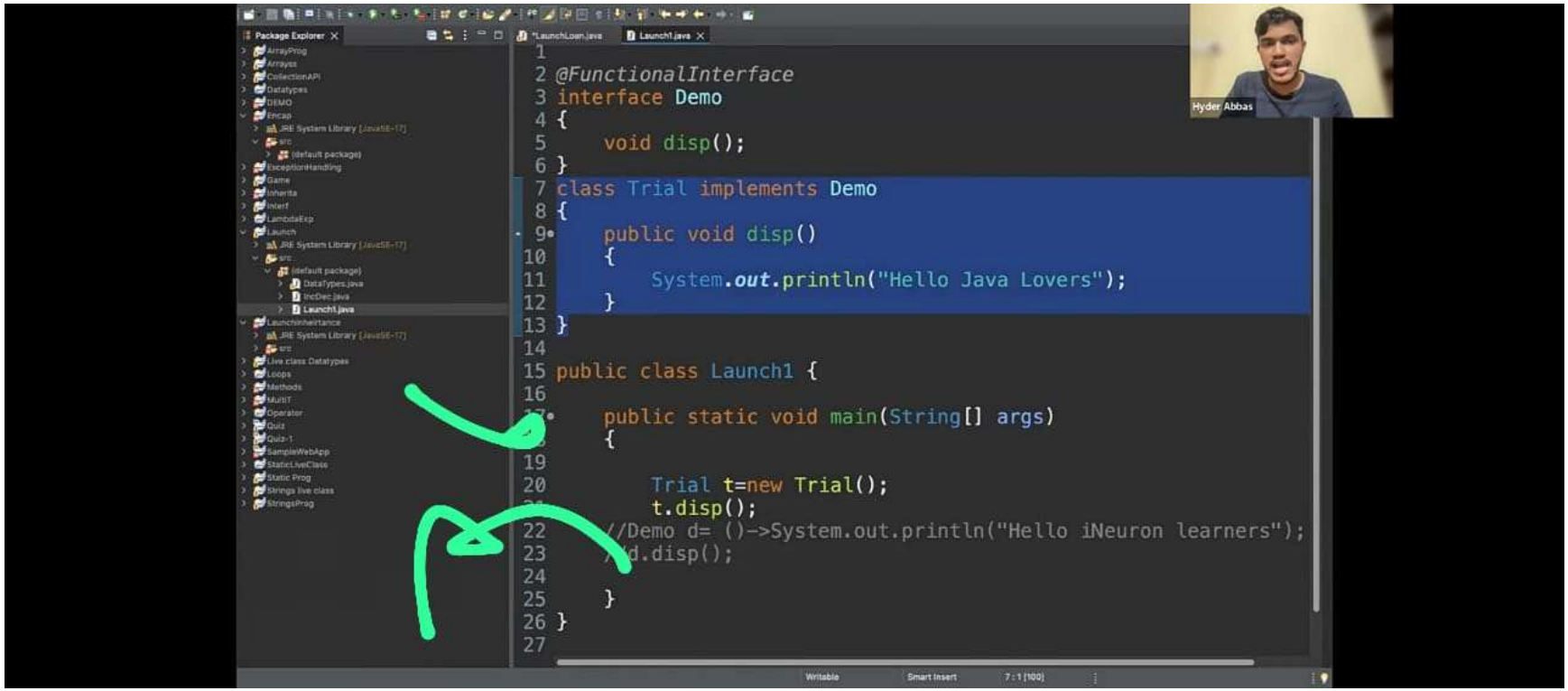

keep.google.com/#NOTE/1668526261311.389138131
=> Lambda expression => functional interface => interface
abstract
+
inheritance
Hyder Abbas

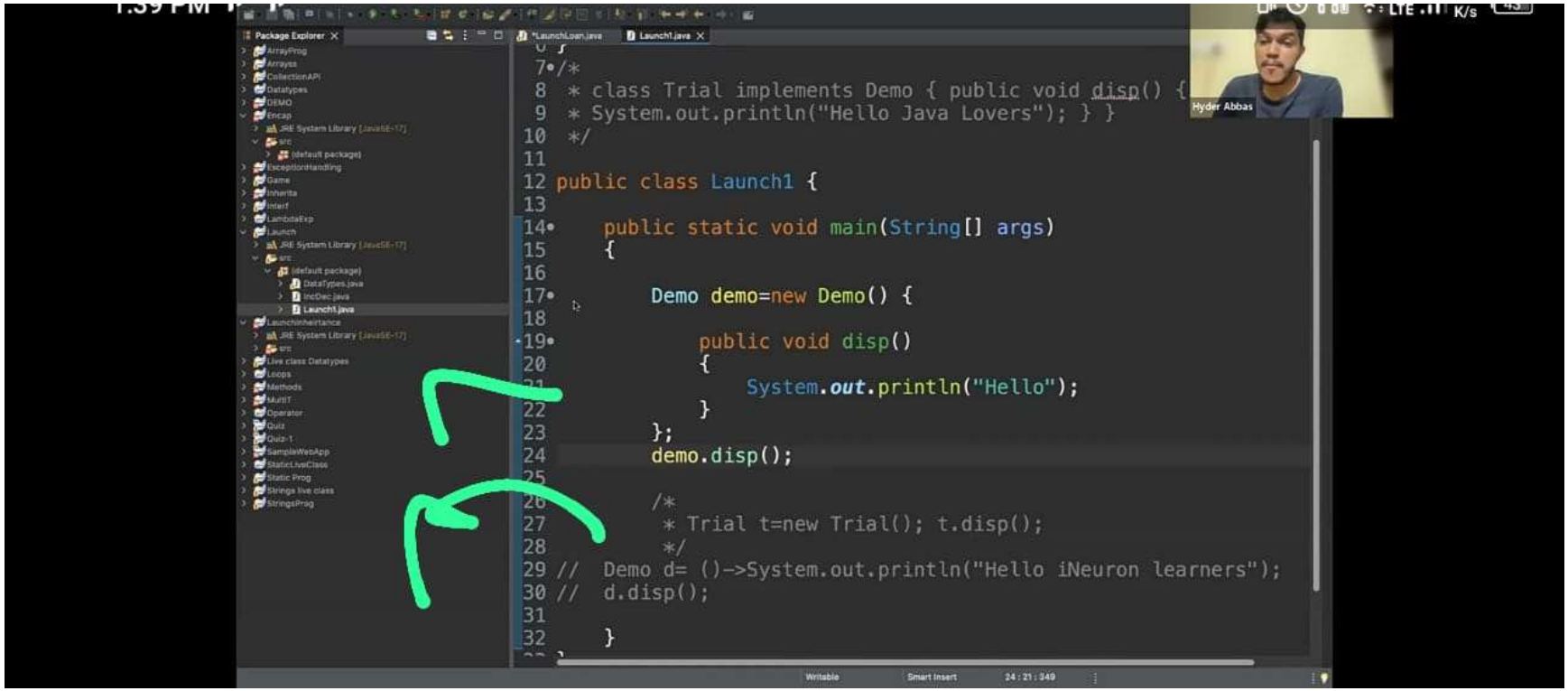

Inbox (364) - syedhyder@
Hyder Abbas - Jira Service
Google Keep
Google Keep
Learn Dashboard | iNeuron
Google Keep
keep.google.com/#NOTE/1668526261311.389138131
Void disp( ) parathi
s.o.p ("Hello");
Body
arrow operator
Lambda operator
To write
Lambda expression
=>
( ) -> {
s.o.p("Hello");
}
Hyder Abbas

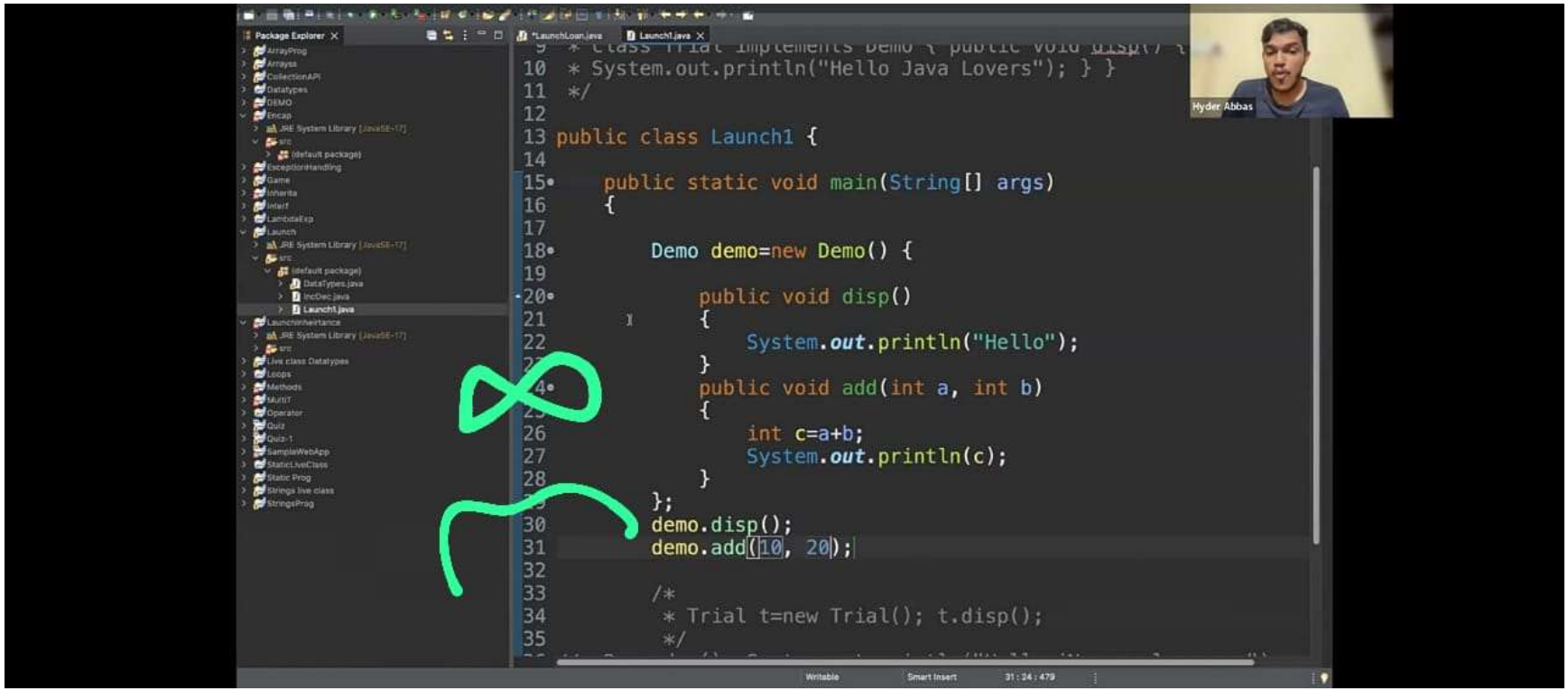

Inbox (364) - syedhyder@
Hyder Abbas - Jira Service
Google Keep
Google Keep
Learn Dashboard | iNeuron
Google Keep
keep.google.com/#NOTE/1668526261311.389138131
Hyder Abbas
① we
→
② 2 parts
↳ left of lambda operator parameters
↳ right of lambda operator body
void disp()
{ s.o.p(hello);
}
() -> {
s.o.p(Hello);
}
+
() -> s.o.p(Hello);

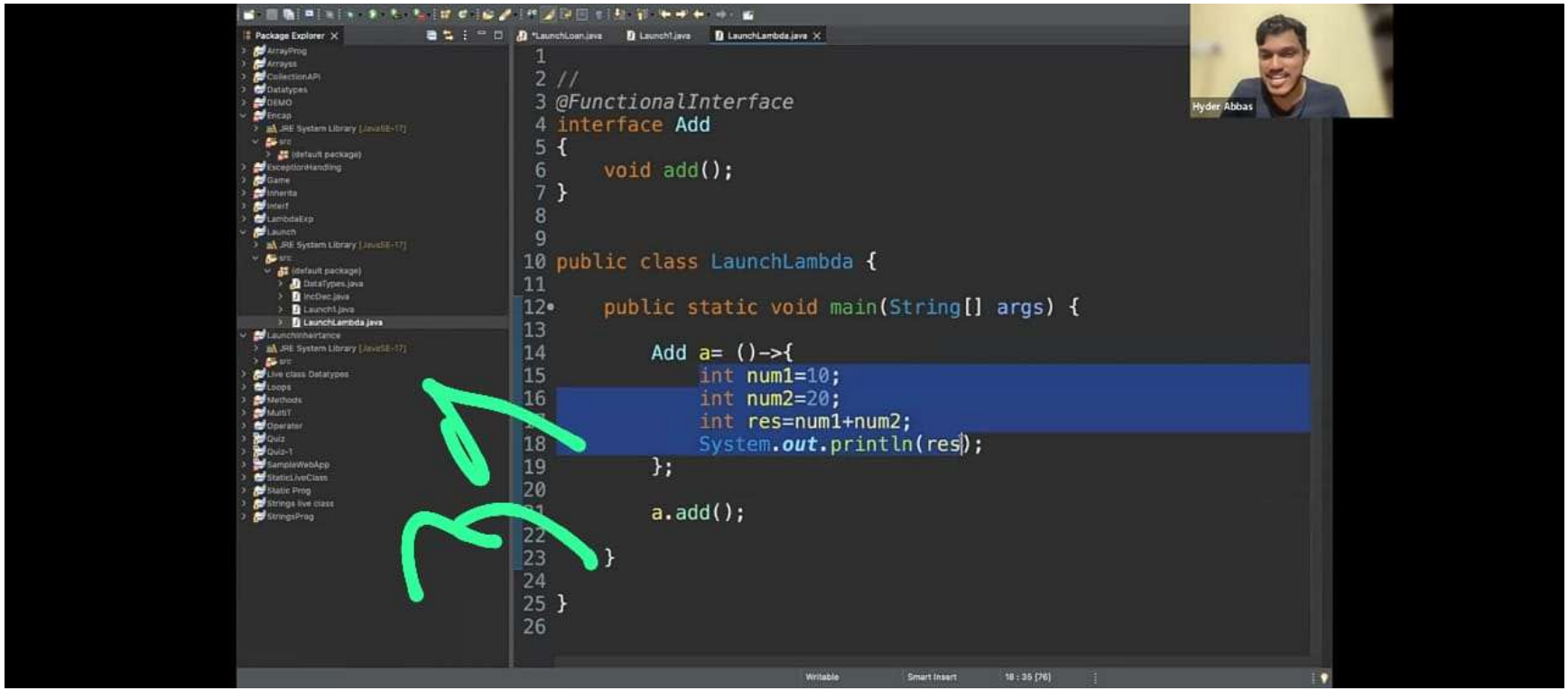

keep.google.com/#NOTE/1668528261311.389138131
Hyder Abbas
right ↳ lambda operator body
void disp()
{ s.o.println("Hello"); }
disp();
( ) -> {
s.o.println("Hello");
}
+
( ) -> s.o.println("Hello");

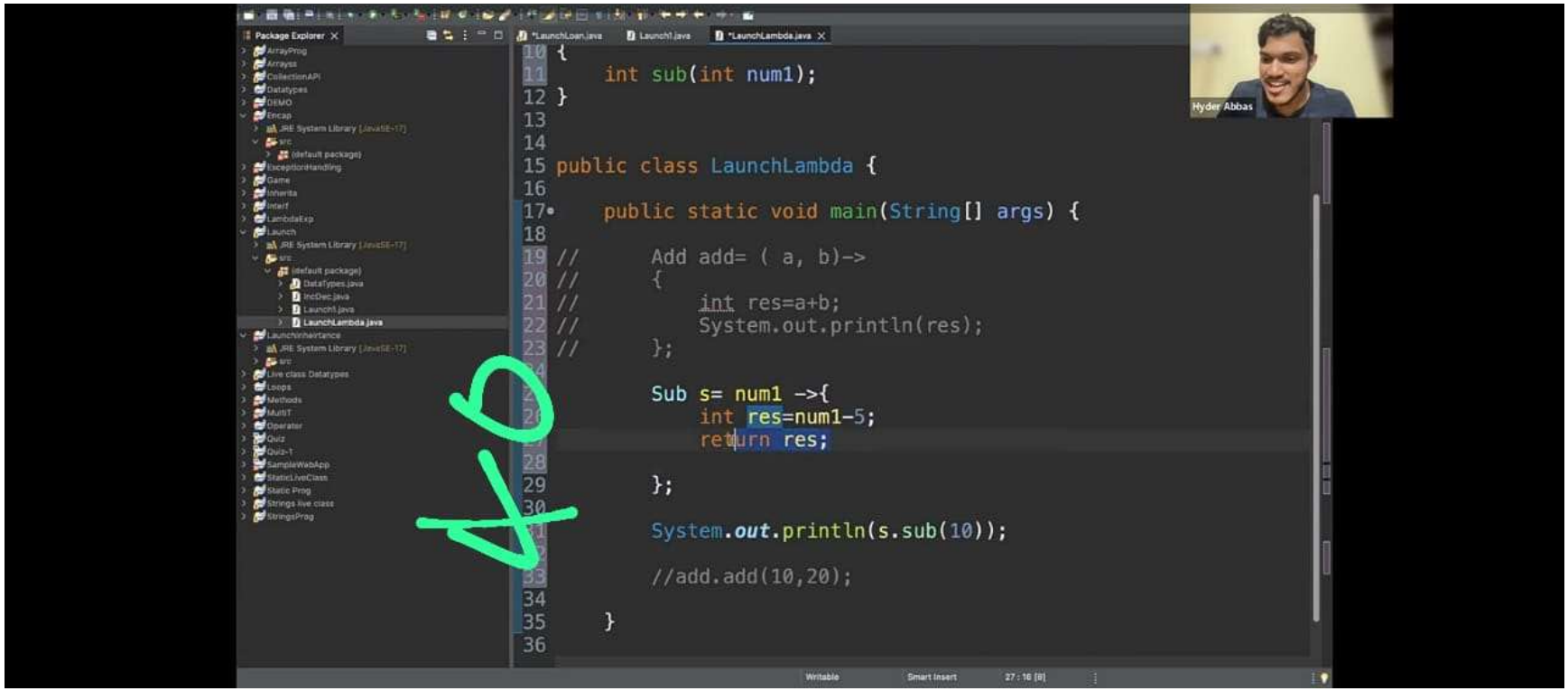

Package Explorer
*LaunchLoan.java
Launch1.java
@FunctionalInterface
interface Demo
{
void disp();
}
public class Launch1 {
public static void main(String[] args)
{
Demo d= ()->
{
System.out.println("Hello iNeuron learners");
};
d.disp();
}
}
Hyder Abbas
Writable
Smart Insert
16 : 15 : 204

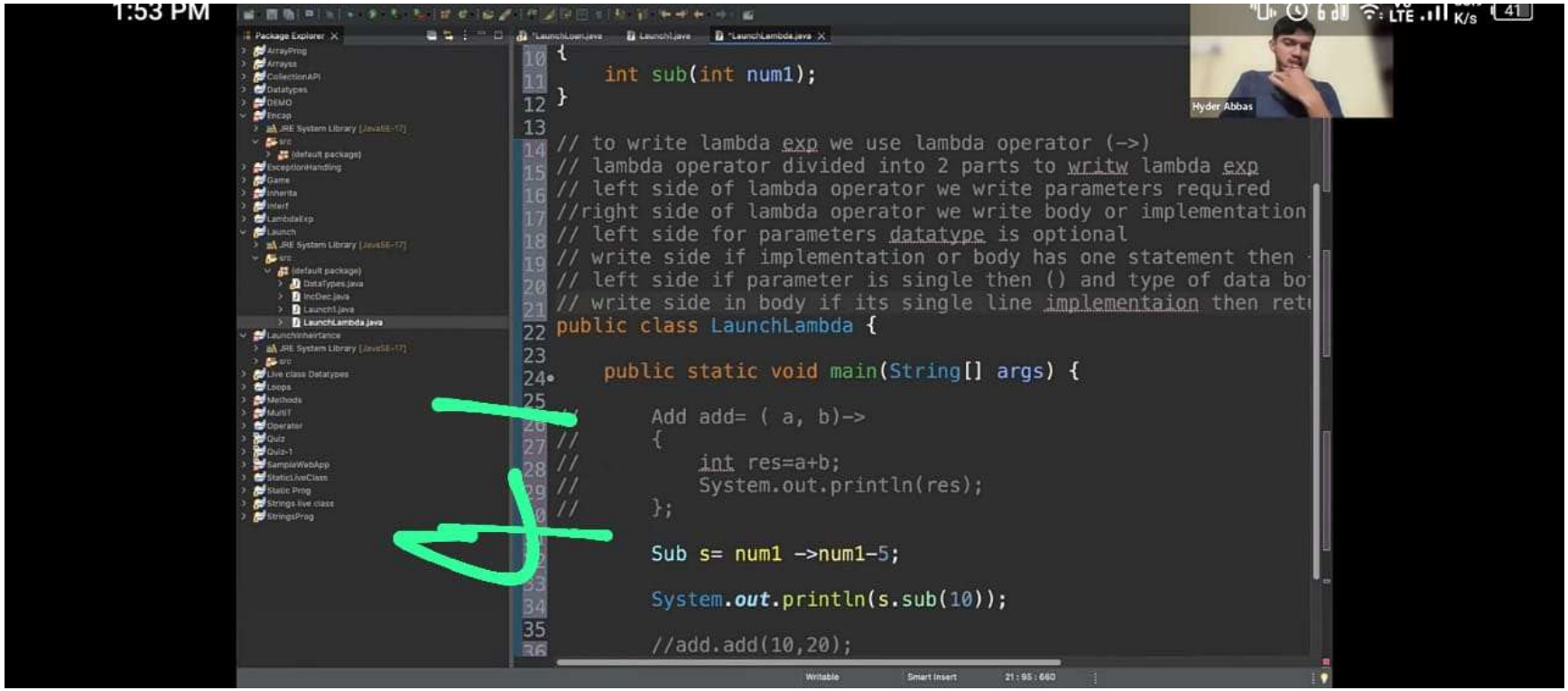

Package Explorer
*LaunchLoan.java
Launch1.java
@FunctionalInterface
interface Demo
{
void disp();
}
class Trial implements Demo
{
public void disp()
{
System.out.println("Hello Java Lovers");
}
}
public class Launch1 {
public static void main(String[] args)
{
Trial t=new Trial();
t.disp();
//Demo d= ()->System.out.println("Hello iNeuron learners");
//d.disp();
}
}
Hyder Abbas
Writable
Smart Insert
7 : 1 [100]

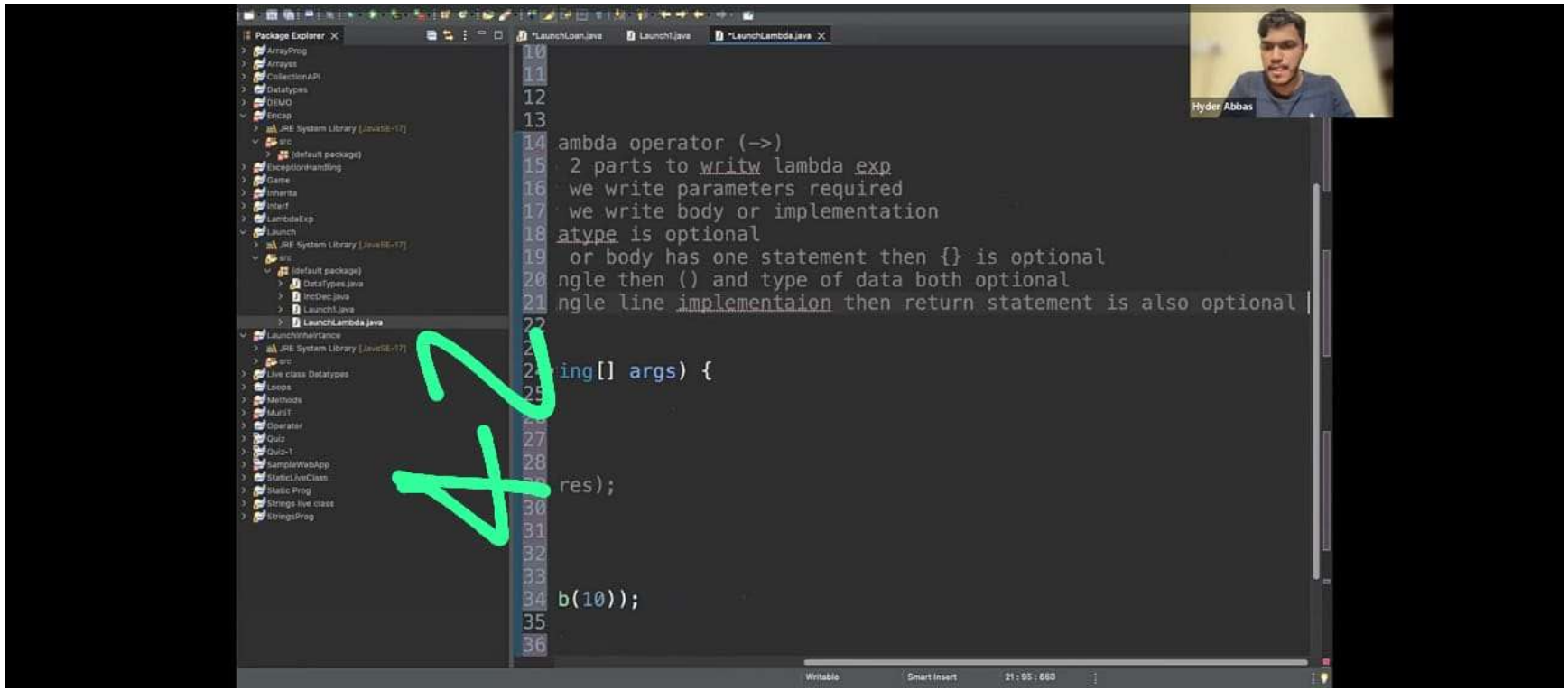

Package Explorer
*LaunchLoan.java
Launch1.java
/*
* class Trial implements Demo { public void disp() {
* System.out.println("Hello Java Lovers"); } }
*/
public class Launch1 {
public static void main(String[] args)
{
Demo demo=new Demo() {
public void disp()
{
System.out.println("Hello");
}
};
demo.disp();
/*
* Trial t=new Trial(); t.disp();
*/
// Demo d= ()->System.out.println("Hello iNeuron learners");
// d.disp();
}
Hyder Abbas
Writable
Smart Insert
24 : 21 : 349

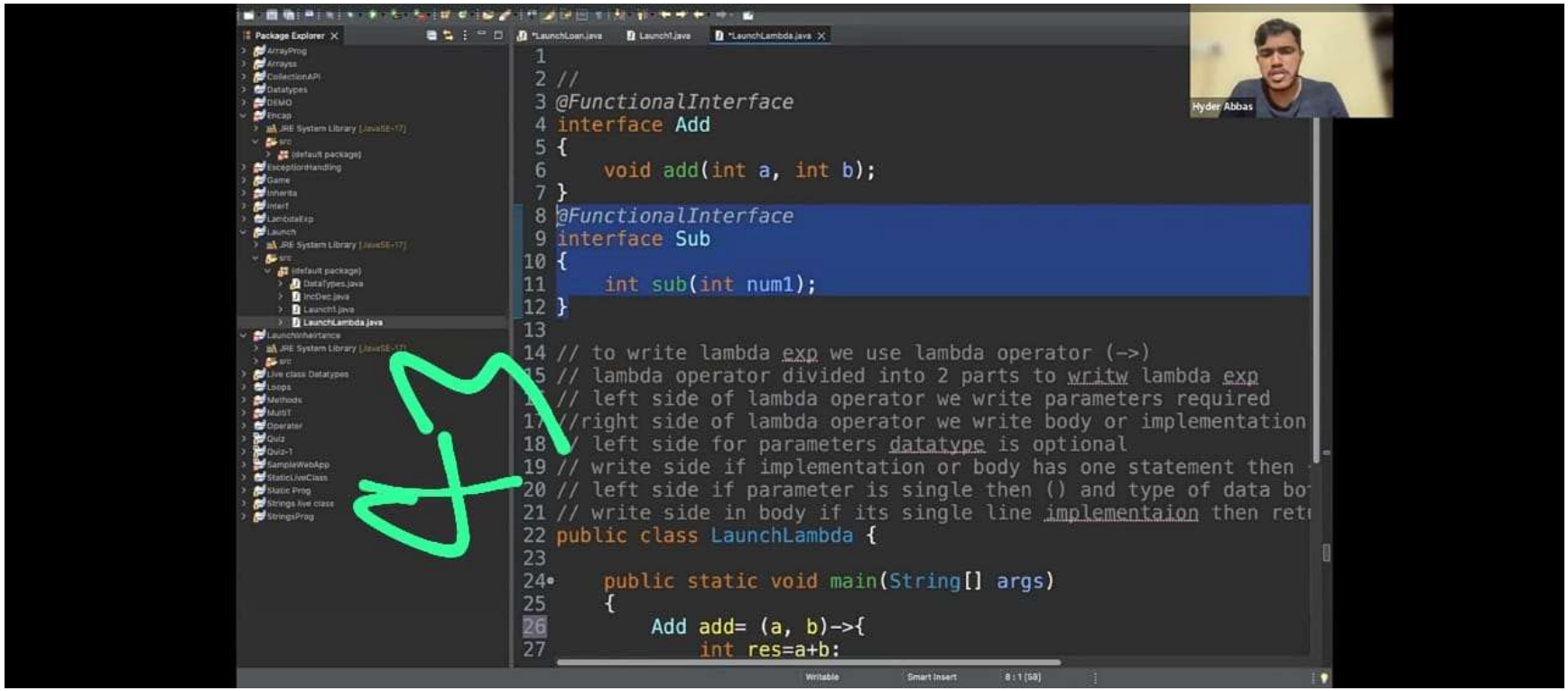

Package Explorer
*LaunchLoan.java
Launch1.java
* System.out.println("Hello Java Lovers"); } }
*/
public class Launch1 {
public static void main(String[] args)
{
Demo demo=new Demo() {
public void disp()
{
System.out.println("Hello");
}
public void add(int a, int b)
{
int c=a+b;
System.out.println(c);
}
};
demo.disp();
demo.add(10, 20);
/*
* Trial t=new Trial(); t.disp();
*/
Hyder Abbas
Writable
Smart Insert
31 : 24 : 479

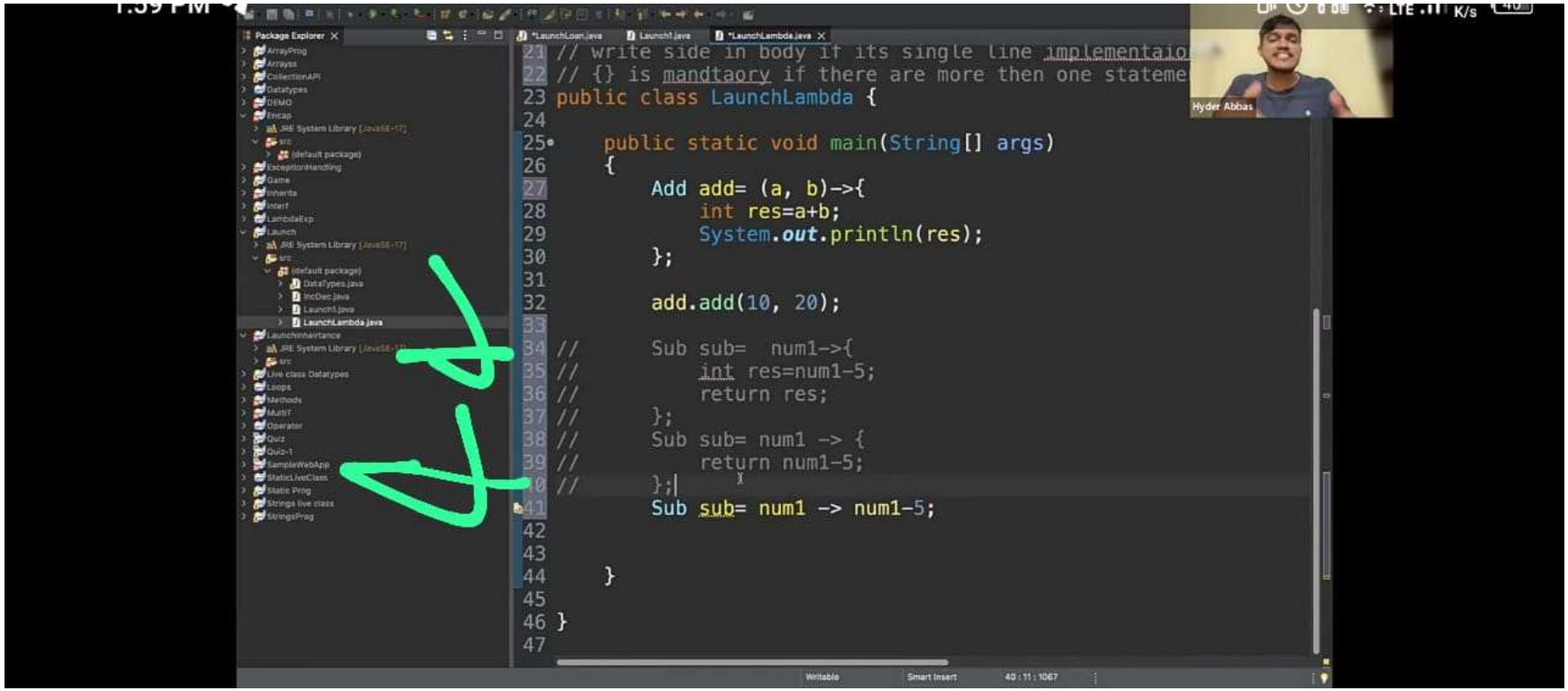

Package Explorer
*LaunchLoan.java
Launch1.java
LaunchLambda.java
Hyder Abbas
//
@FunctionalInterface
interface Add
{
void add();
}
public class LaunchLambda {
public static void main(String[] args) {
Add a= ()->{
int num1=10;
int num2=20;
int res=num1+num2;
System.out.println(res);
};
a.add();
}
}
Writable
Smart Insert
18 : 35 [76]

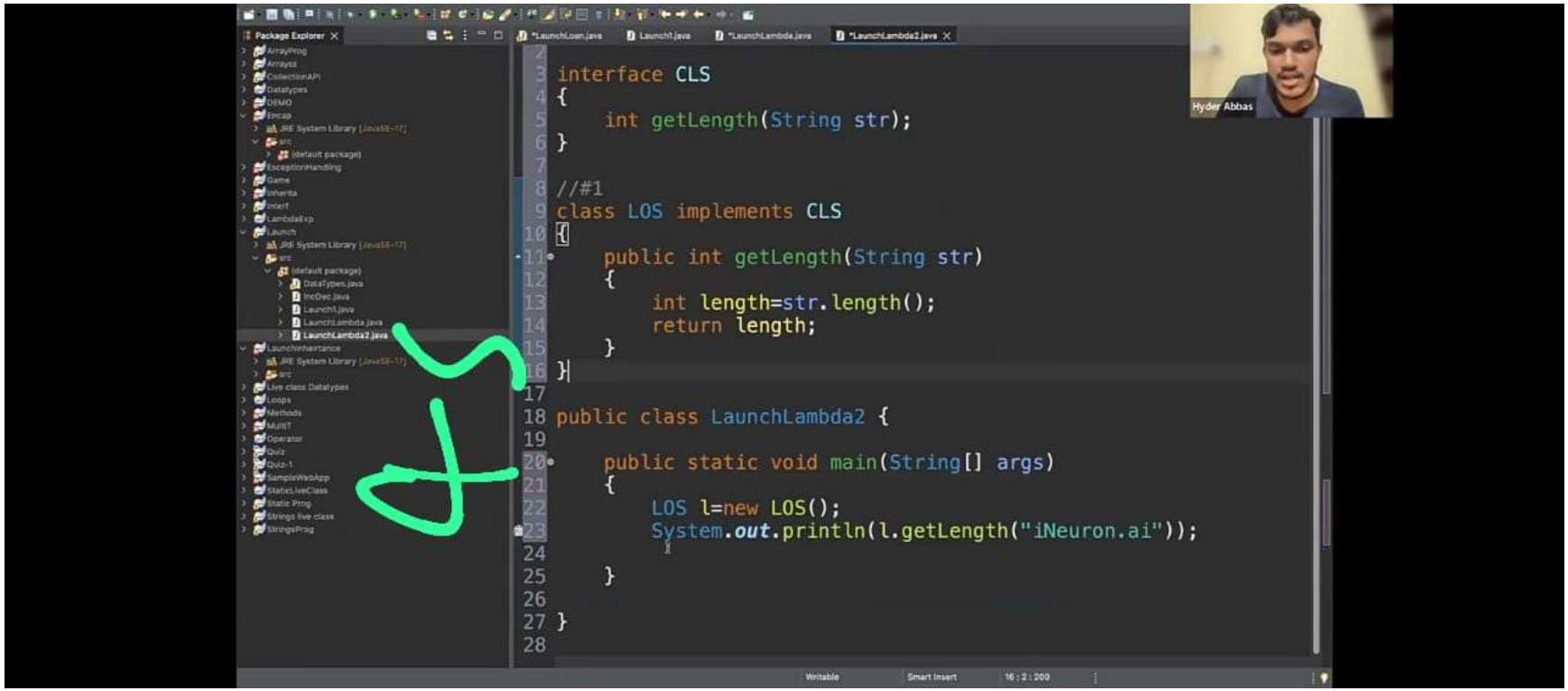

Package Explorer
*LaunchLoan.java
Launch1.java
*LaunchLambda.java
Hyder Abbas
{
int sub(int num1);
}
public class LaunchLambda {
public static void main(String[] args) {
// Add add= (a, b)->
// {
// int res=a+b;
// System.out.println(res);
// };
Sub s= num1 ->{
int res=num1-5;
return res;
};
System.out.println(s.sub(10));
//add.add(10,20);
}
Writable
Smart Insert
27 : 16 [8]

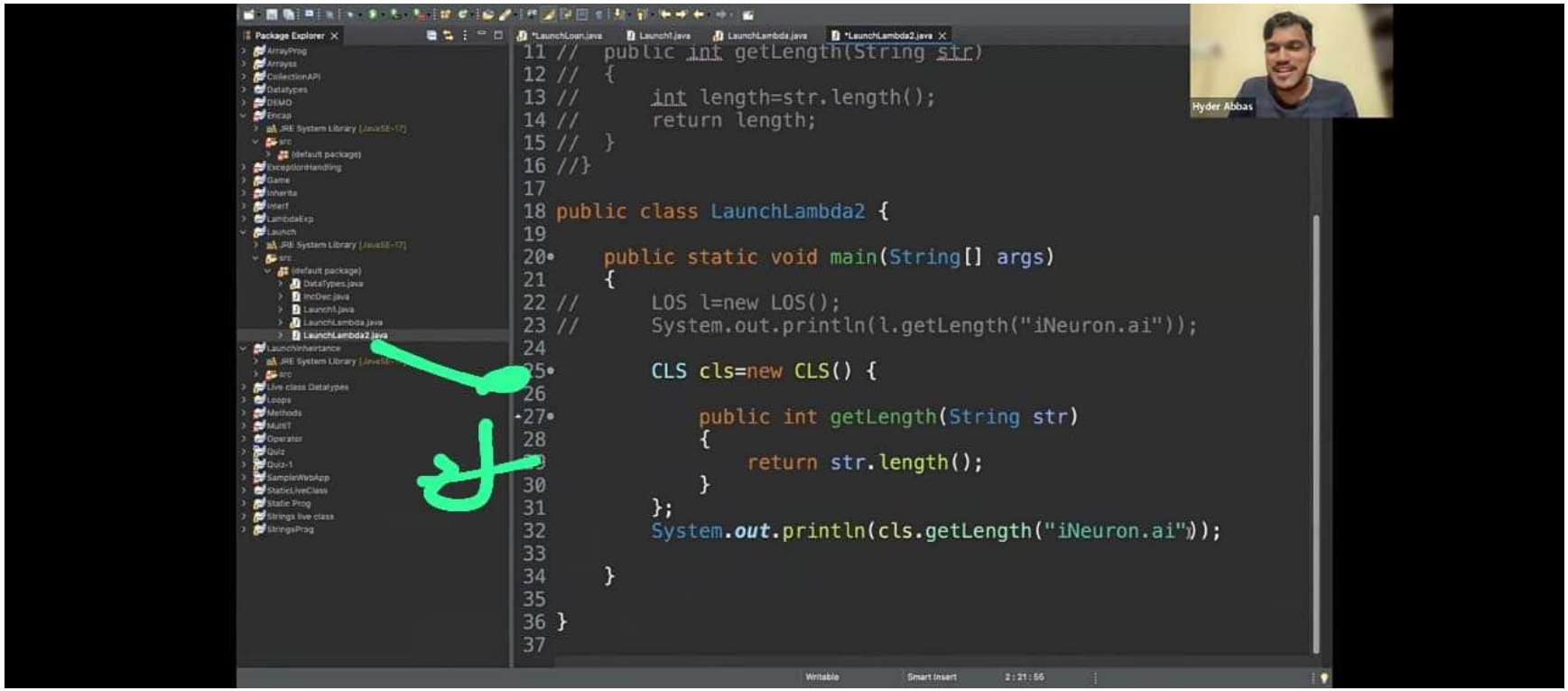

1:53 PM
Hyder Abbas
{
int sub(int num1);
}
// to write lambda exp we use lambda operator (->)
// lambda operator divided into 2 parts to writw lambda exp
// left side of lambda operator we write parameters required
//right side of lambda operator we write body or implementation
// left side for parameters datatype is optional
// write side if implementation or body has one statement then
// left side if parameter is single then () and type of data bo
// write side in body if its single line implementaion then ret
public class LaunchLambda {
public static void main(String[] args) {
Add add= (a, b)->
{
int res=a+b;
System.out.println(res);
};
Sub s= num1 ->num1-5;
System.out.println(s.sub(10));
//add.add(10,20);
Writable
Smart Insert
21 : 95 : 660

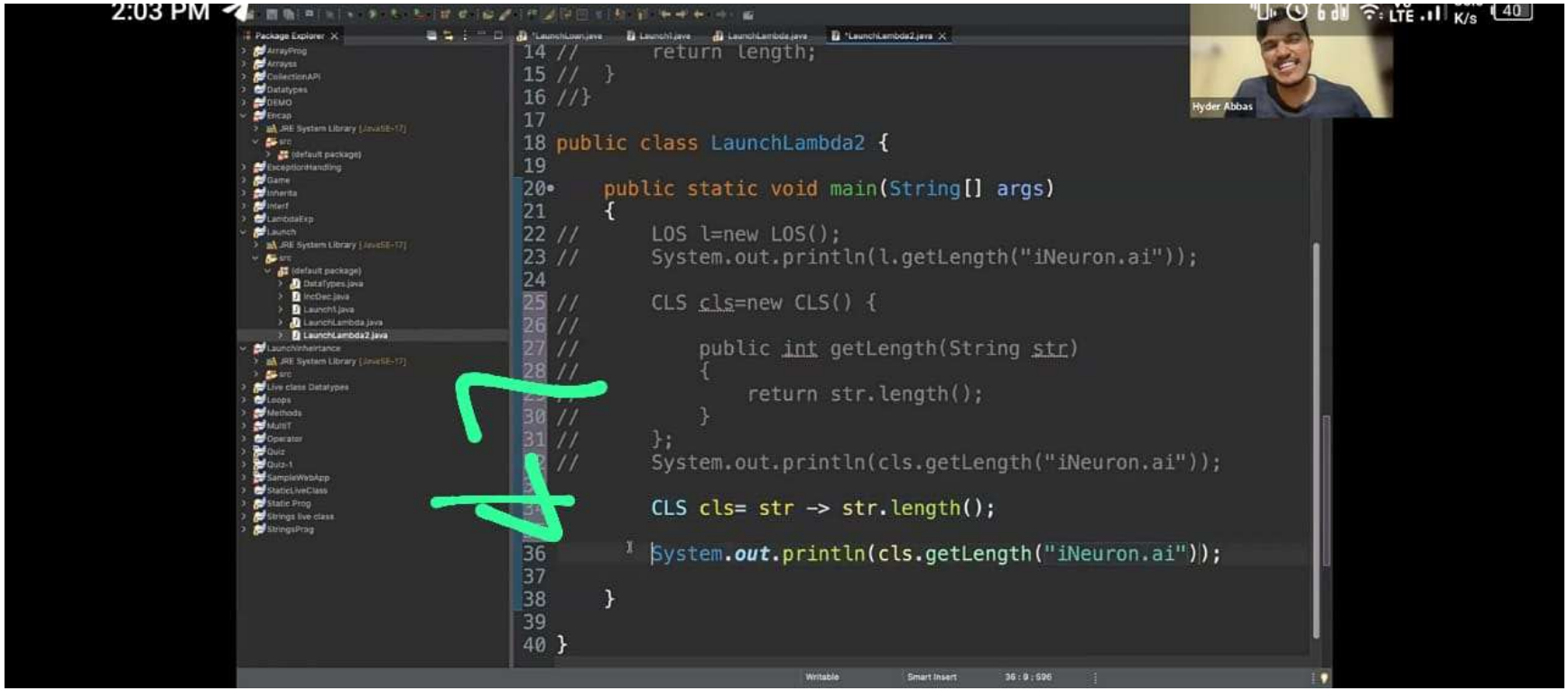

Package Explorer
*LaunchLoan.java
Launch1.java
*LaunchLambda.java
ambda operator (->)
2 parts to writw lambda exp
we write parameters required
we write body or implementation
atype is optional
or body has one statement then {} is optional
ngle then () and type of data both optional
ngle line implementaion then return statement is also optional
ing[] args) {
res);
b(10));
Hyder Abbas
Writable
Smart Insert
21 : 95 : 660


Package Explorer
*LaunchLoan.java
Launch1.java
*LaunchLambda.java
Hyder Abbas
//
@FunctionalInterface
interface Add
{
void add(int a, int b);
}
@FunctionalInterface
interface Sub
{
int sub(int num1);
}
// to write lambda exp we use lambda operator (->)
// lambda operator divided into 2 parts to writw lambda exp
// left side of lambda operator we write parameters required
//right side of lambda operator we write body or implementation
// left side for parameters datatype is optional
// write side if implementation or body has one statement then
// left side if parameter is single then () and type of data bo
// write side in body if its single line implementaion then ret
public class LaunchLambda {
public static void main(String[] args)
{
Add add= (a, b)->{
int res=a+b;
Writable
Smart Insert
8 : 1 (58)


Package Explorer
*LaunchLoan.java
Launch1.java
*LaunchLambda.java
// write side in body if its single line implementaio
// {} is mandtaory if there are more then one stateme
public class LaunchLambda {
public static void main(String[] args)
{
Add add= (a, b)->{
int res=a+b;
System.out.println(res);
};
add.add(10, 20);
// Sub sub= num1->{
// int res=num1-5;
// return res;
// };
// Sub sub= num1 -> {
// return num1-5;
// };
Sub sub= num1 -> num1-5;
}
}
Hyder Abbas
Writable
Smart Insert
40 : 11 : 1067


Package Explorer
*LaunchLoan.java
Launch1.java
*LaunchLambda.java
*LaunchLambda2.java
interface CLS
{
int getLength(String str);
}
//#1
class LOS implements CLS
{
public int getLength(String str)
{
int length=str.length();
return length;
}
}
public class LaunchLambda2 {
public static void main(String[] args)
{
LOS l=new LOS();
System.out.println(l.getLength("iNeuron.ai"));
}
}
Hyder Abbas
Writable
Smart Insert
16 : 2 : 200


Package Explorer
*LaunchLoan.java
Launch1.java
LaunchLambda.java
*LaunchLambda2.java
Hyder Abbas
11 // public int getLength(String str)
12 // {
13 //   int length=str.length();
14 //   return length;
15 // }
16 //}
17
18 public class LaunchLambda2 {
19
20•   public static void main(String[] args)
21   {
22 //   LOS l=new LOS();
23 //   System.out.println(l.getLength("iNeuron.ai"));
24
25•   CLS cls=new CLS() {
26
27•   public int getLength(String str)
28   {
30   return str.length();
30   }
31   };
32   System.out.println(cls.getLength("iNeuron.ai"));
33
34   }
35
36 }
37
Writable
Smart Insert
2 : 21 : 55


2:03 PM
Hyder Abbas
14 //		return length;
15 //	}
16 //}
17
18 public class LaunchLambda2 {
19
20	public static void main(String[] args)
21	{
22 //		LOS l=new LOS();
23 //		System.out.println(l.getLength("iNeuron.ai"));
24
25 //		CLS cls=new CLS() {
26 //
27 //			public int getLength(String str)
28 //			{
29 //				return str.length();
30 //			}
31 //		};
32 //		System.out.println(cls.getLength("iNeuron.ai"));
33
34		CLS cls= str -> str.length();
35
36		System.out.println(cls.getLength("iNeuron.ai"));
37
38	}
39
40 }
Writable
Smart Insert
36 : 9 : 596