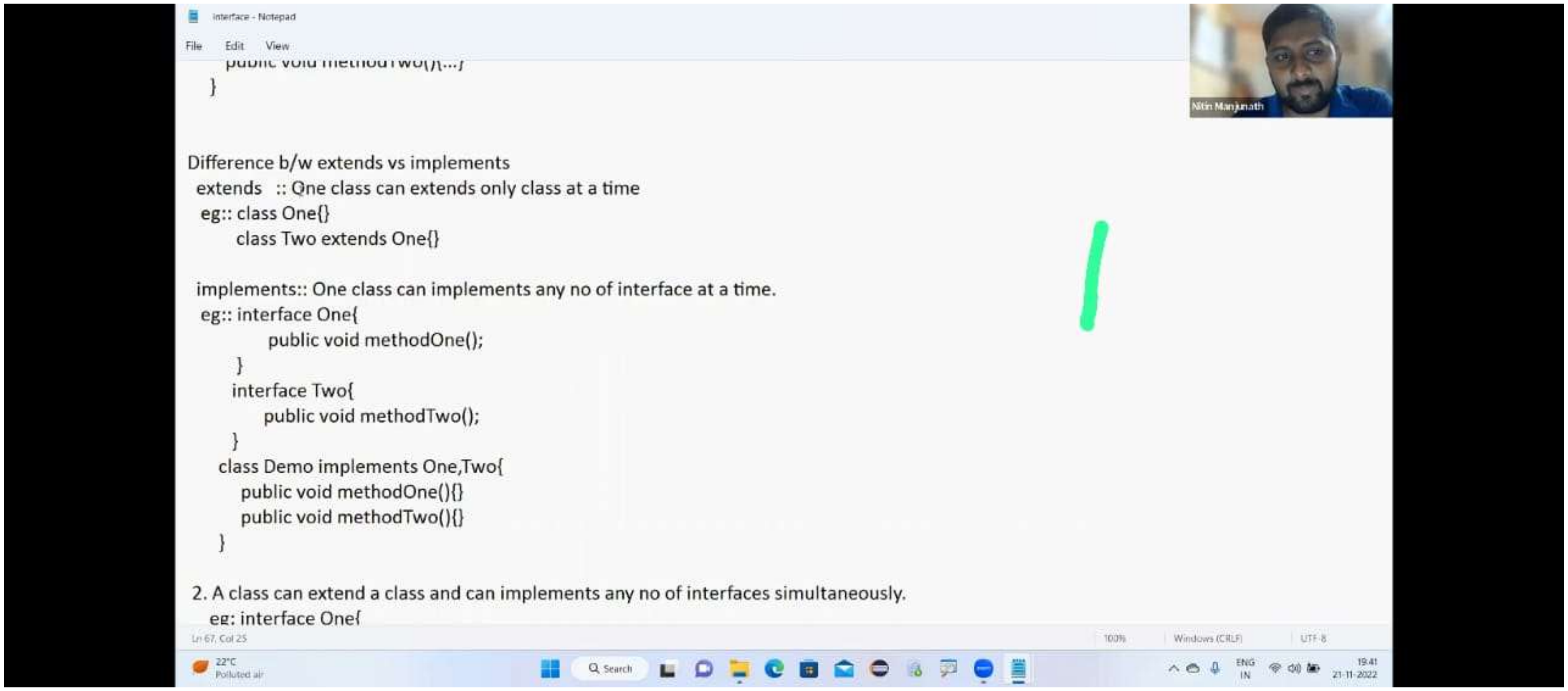
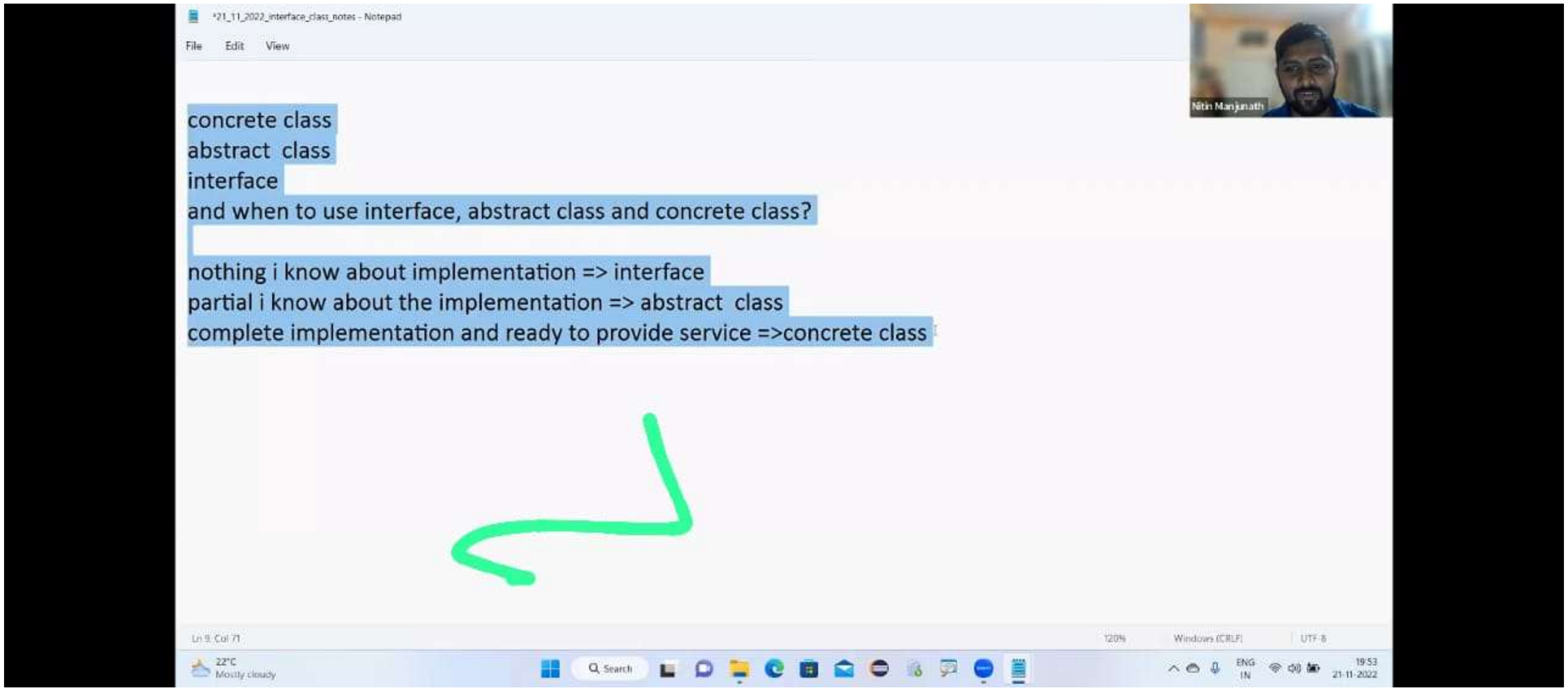
Java Interfaces Part3

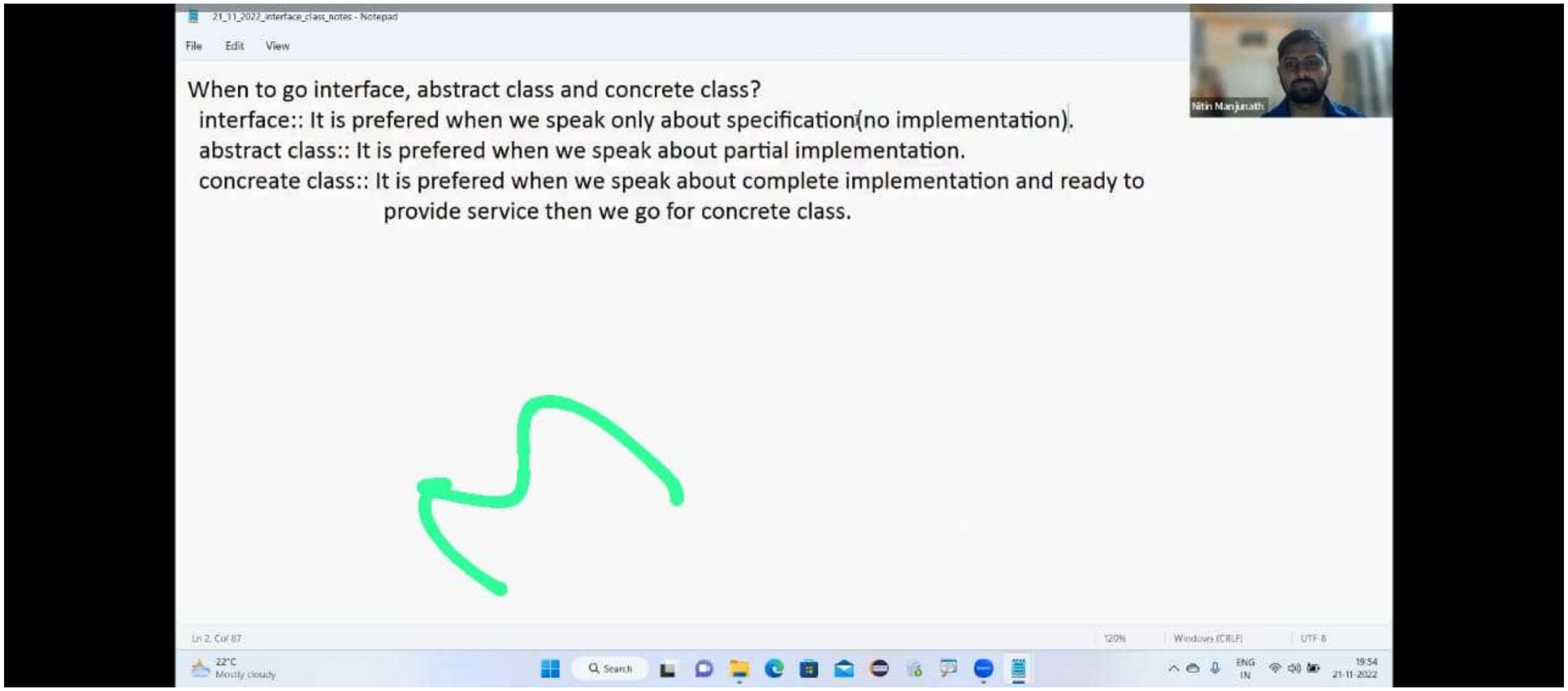


```
    public void methodTwo(){...}
  }


Difference b/w extends vs implements
 extends   :: One class can extends only class at a time
  eg:: class One{}
        class Two extends One{}

 implements:: One class can implements any no of interface at a time.
  eg:: interface One{
             public void methodOne();
       }
      interface Two{
            public void methodTwo();
      }
    class Demo implements One,Two{
        public void methodOne(){}
        public void methodTwo(){}
    }

2. A class can extend a class and can implements any no of interfaces simultaneously.
   eg: interface One{
```

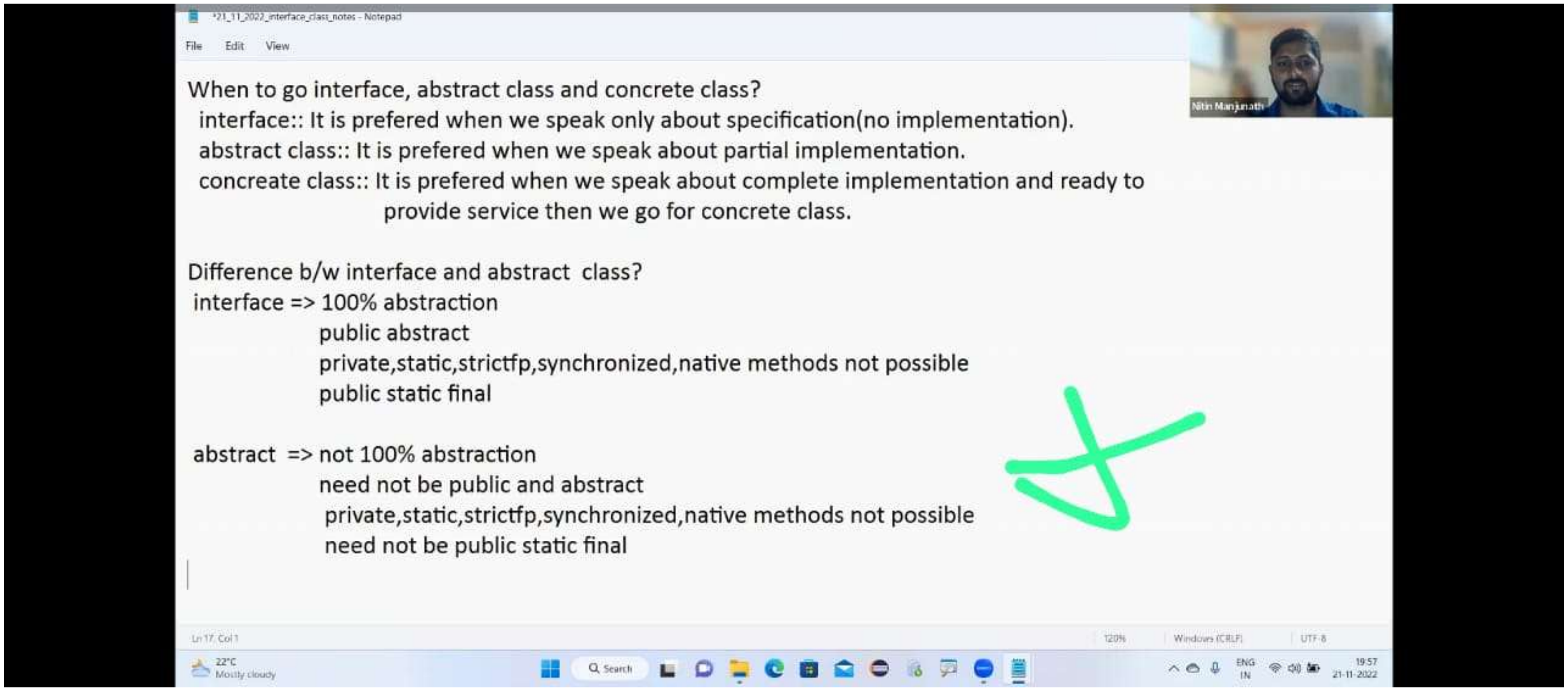


concrete class
abstract class
interface
and when to use interface, abstract class and concrete class?

nothing i know about implementation => interface
partial i know about the implementation => abstract class
complete implementation and ready to provide service =>concrete class

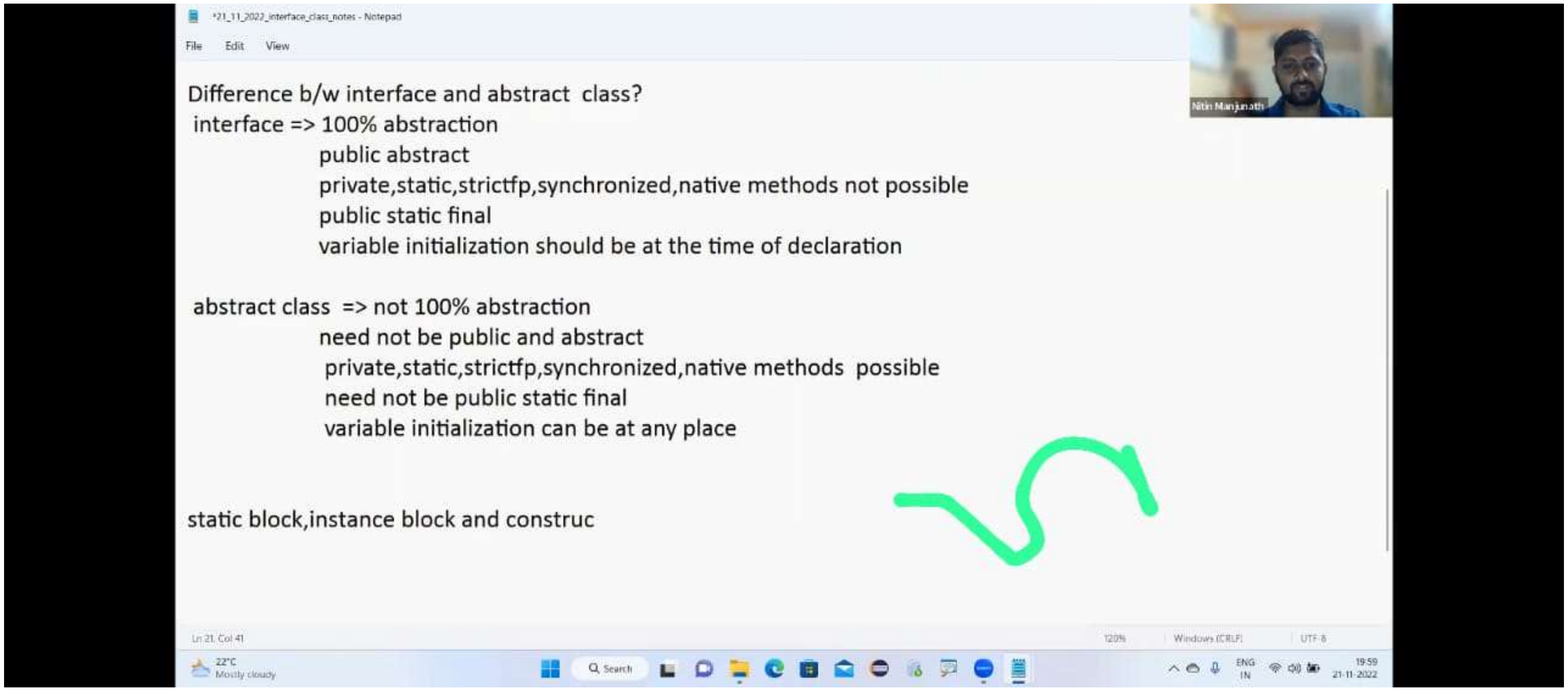


```
When to go interface, abstract class and concrete class?
 interface:: It is prefered when we speak only about specification(no implementation).
 abstract class:: It is prefered when we speak about partial implementation.
 concreate class:: It is prefered when we speak about complete implementation and ready to
                    provide service then we go for concrete class.
```

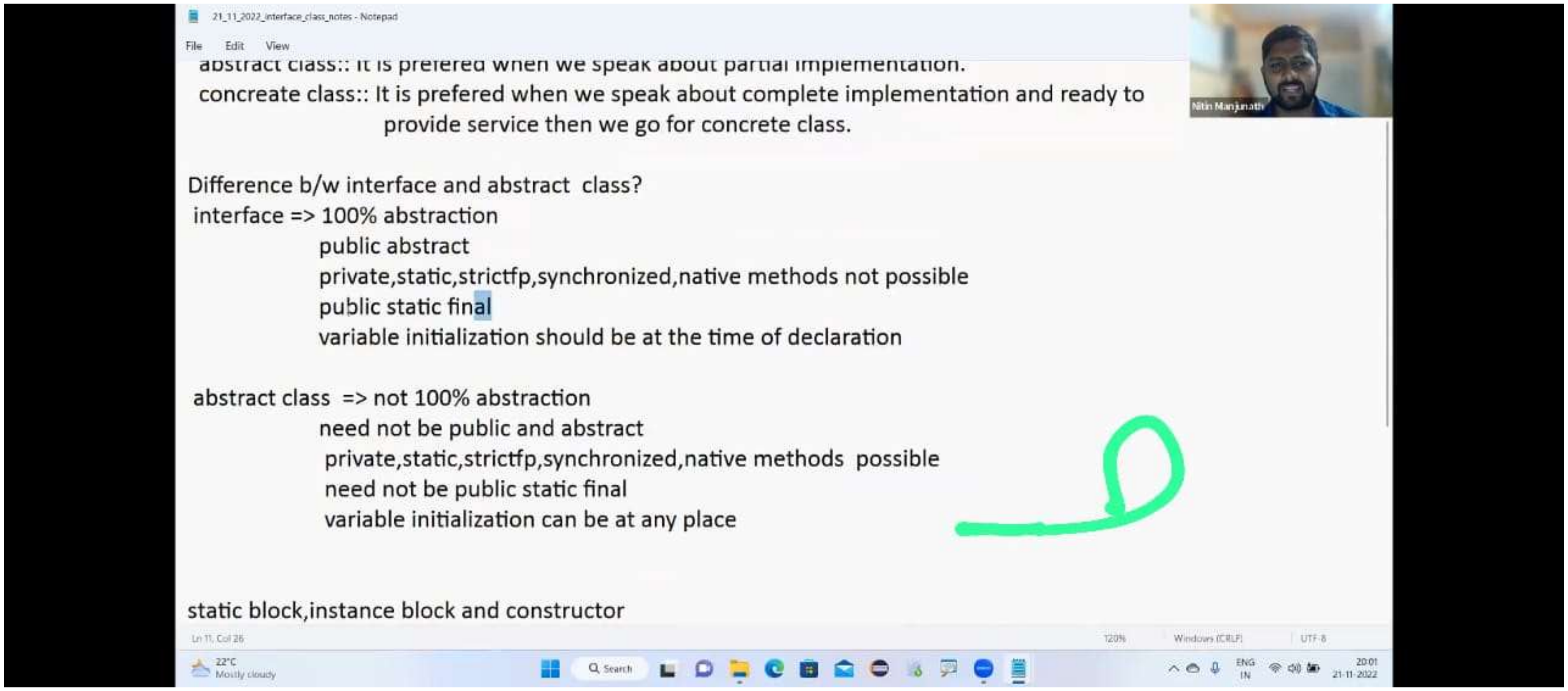


```
When to go interface, abstract class and concrete class?
 interface:: It is prefered when we speak only about specification(no implementation).
 abstract class:: It is prefered when we speak about partial implementation.
 concreate class:: It is prefered when we speak about complete implementation and ready to
                   provide service then we go for concrete class.

Difference b/w interface and abstract  class?
 interface => 100% abstraction
              public abstract
              private,static,strictfp,synchronized,native methods not possible
              public static final

 abstract  => not 100% abstraction
              need not be public and abstract
               private,static,strictfp,synchronized,native methods not possible
               need not be public static final
```

4

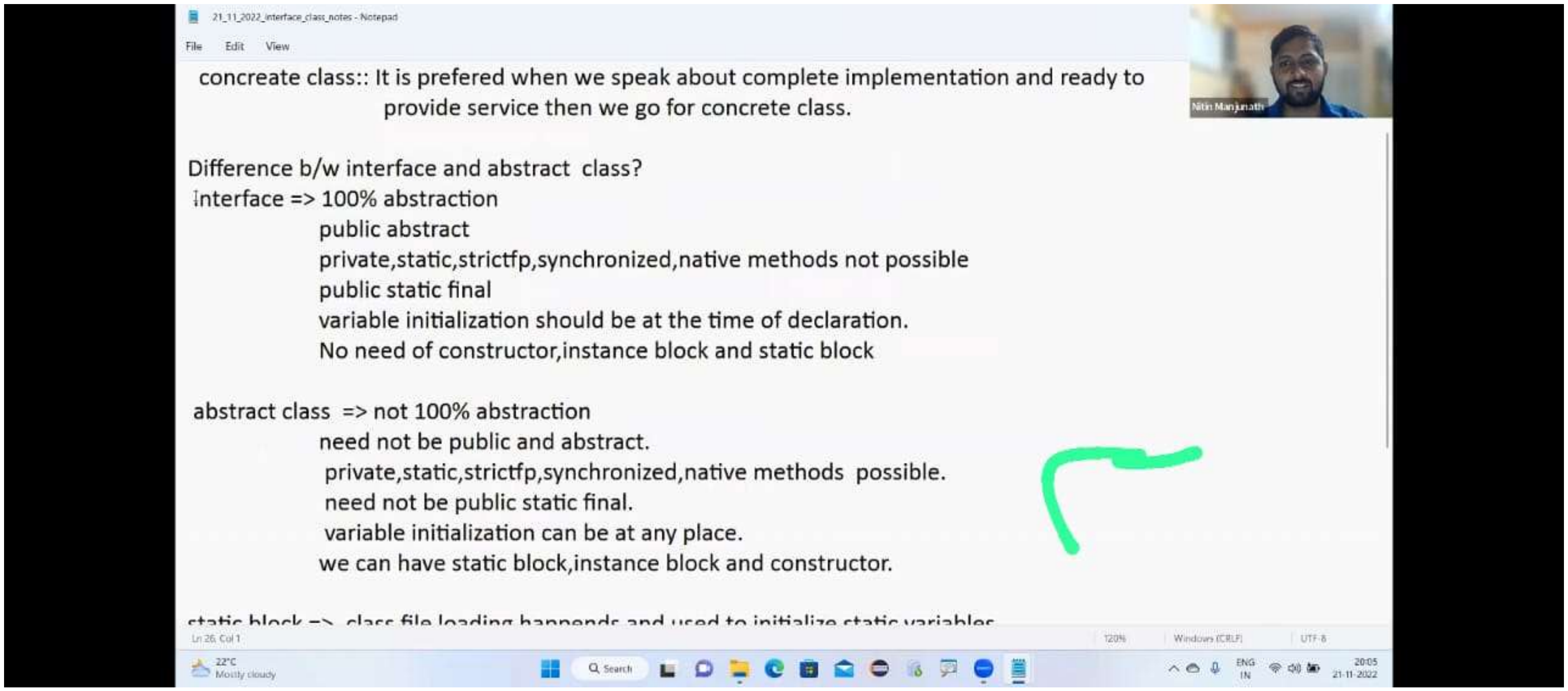


```
Difference b/w interface and abstract  class?
 interface => 100% abstraction
                 public abstract
                 private,static,strictfp,synchronized,native methods not possible
                 public static final
                 variable initialization should be at the time of declaration

 abstract class  => not 100% abstraction
                 need not be public and abstract
                  private,static,strictfp,synchronized,native methods  possible
                  need not be public static final
                  variable initialization can be at any place


static block,instance block and construc
```



abstract class:: It is prefered when we speak about partial implementation.
concreate class:: It is prefered when we speak about complete implementation and ready to
provide service then we go for concrete class.

```
Difference b/w interface and abstract  class?
 interface => 100% abstraction
                  public abstract
                  private,static,strictfp,synchronized,native methods not possible
                  public static final
                  variable initialization should be at the time of declaration

 abstract class  => not 100% abstraction
                  need not be public and abstract
                   private,static,strictfp,synchronized,native methods  possible
                   need not be public static final
                   variable initialization can be at any place


static block,instance block and constructor
```



21_11_2022_interface_class_notes - Notepad

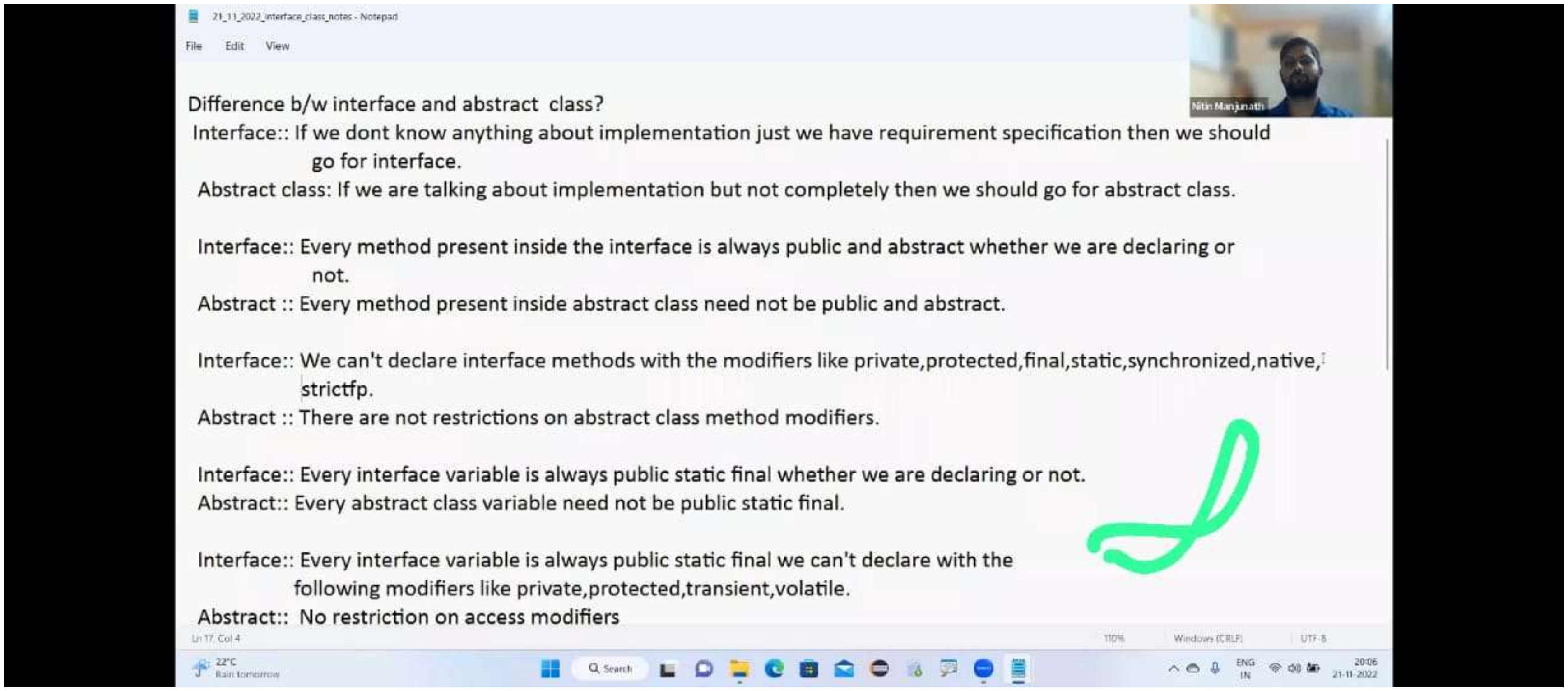
concreate class:: It is prefered when we speak about complete implementation and ready to
provide service then we go for concrete class.

Difference b/w interface and abstract class?

interface => 100% abstraction
- public abstract
- private,static,strictfp,synchronized,native methods not possible
- public static final
- variable initialization should be at the time of declaration.
- No need of constructor,instance block and static block

abstract class => not 100% abstraction
- need not be public and abstract.
- private,static,strictfp,synchronized,native methods possible.
- need not be public static final.
- variable initialization can be at any place.
- we can have static block,instance block and constructor.

static block => class file loading happends and used to initialize static variables



Difference b/w interface and abstract class?

Interface:: If we dont know anything about implementation just we have requirement specification then we should go for interface.

Abstract class: If we are talking about implementation but not completely then we should go for abstract class.

Interface:: Every method present inside the interface is always public and abstract whether we are declaring or not.

Abstract :: Every method present inside abstract class need not be public and abstract.

Interface:: We can't declare interface methods with the modifiers like private,protected,final,static,synchronized,native, strictfp.

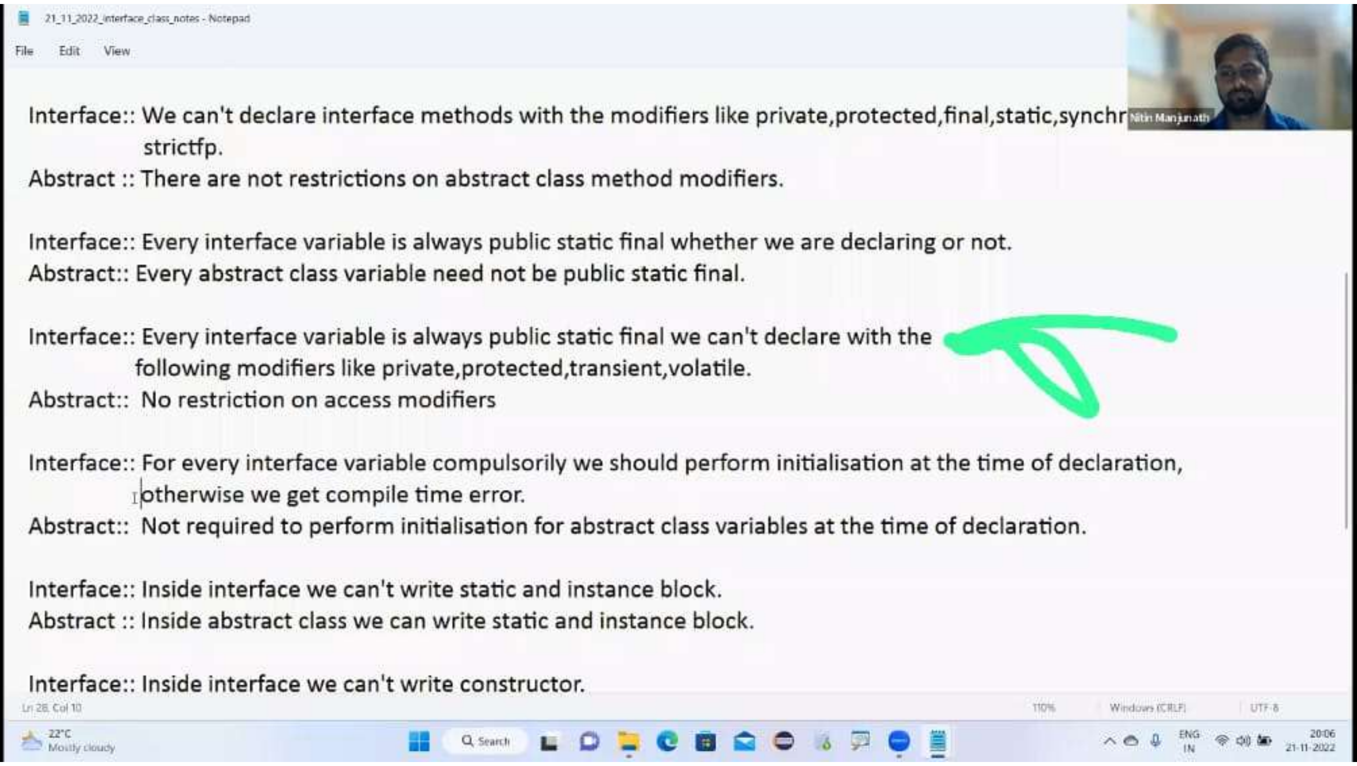
Abstract :: There are not restrictions on abstract class method modifiers.

Interface:: Every interface variable is always public static final whether we are declaring or not.

Abstract:: Every abstract class variable need not be public static final.

Interface:: Every interface variable is always public static final we can't declare with the following modifiers like private,protected,transient,volatile.

Abstract:: No restriction on access modifiers



Interface:: We can't declare interface methods with the modifiers like private,protected,final,static,synchr
strictfp.
Abstract :: There are not restrictions on abstract class method modifiers.

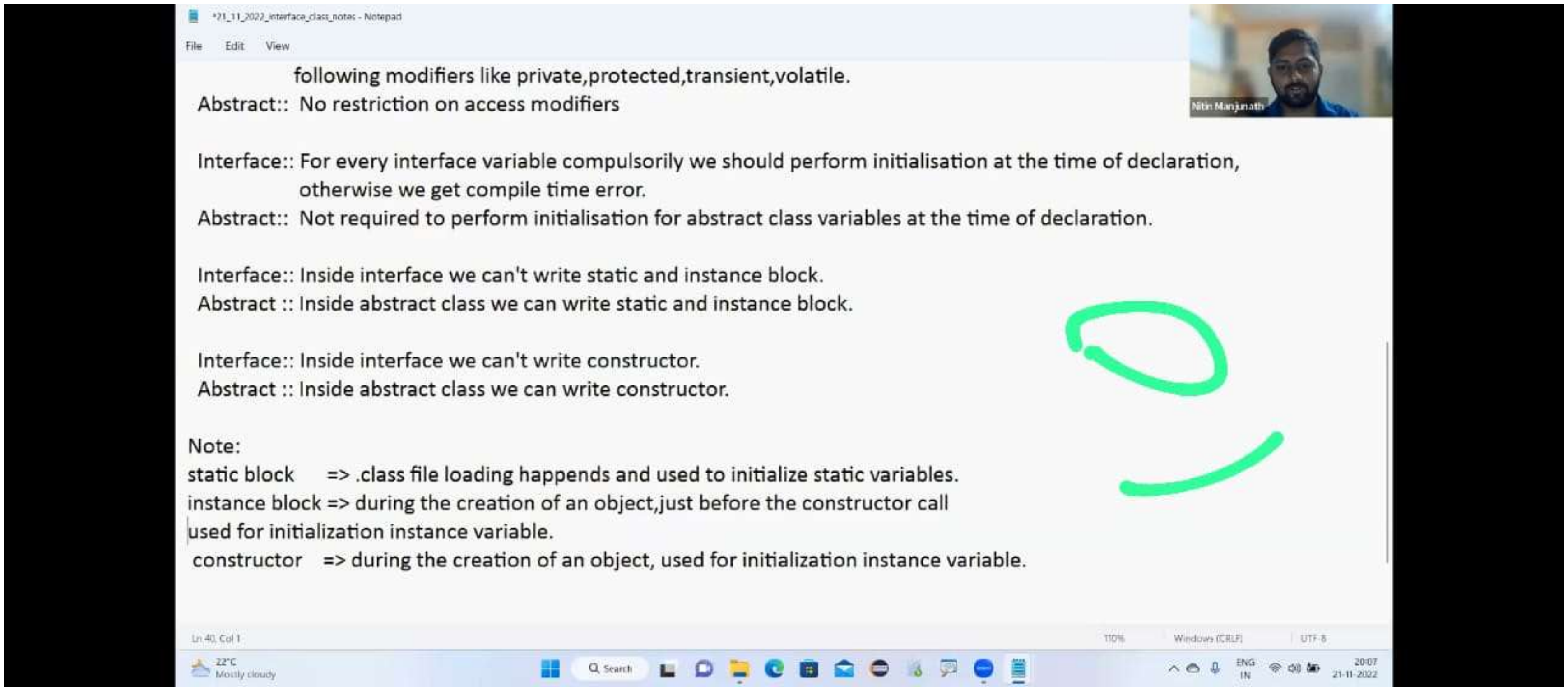
Interface:: Every interface variable is always public static final whether we are declaring or not.
Abstract:: Every abstract class variable need not be public static final.

Interface:: Every interface variable is always public static final we can't declare with the
following modifiers like private,protected,transient,volatile.
Abstract:: No restriction on access modifiers

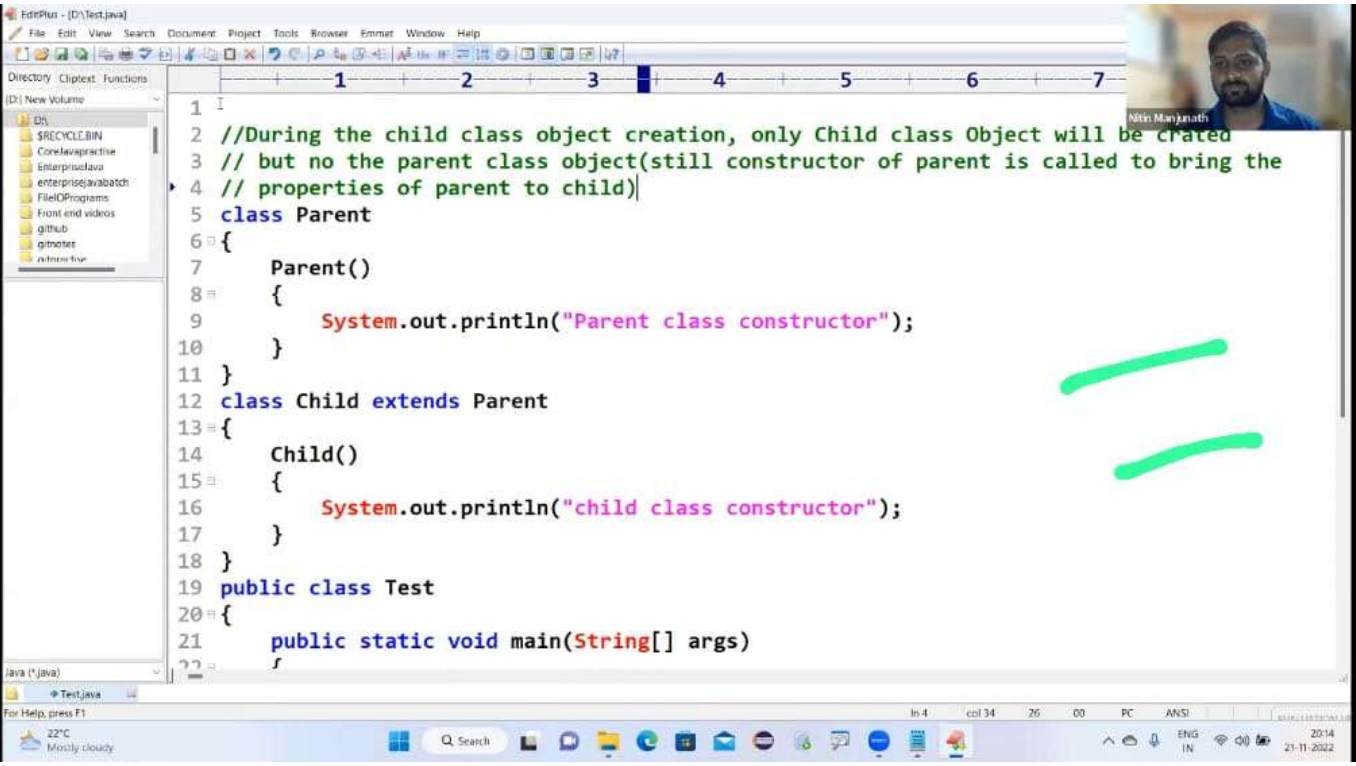
Interface:: For every interface variable compulsorily we should perform initialisation at the time of declaration,
otherwise we get compile time error.
Abstract:: Not required to perform initialisation for abstract class variables at the time of declaration.

Interface:: Inside interface we can't write static and instance block.
Abstract :: Inside abstract class we can write static and instance block.

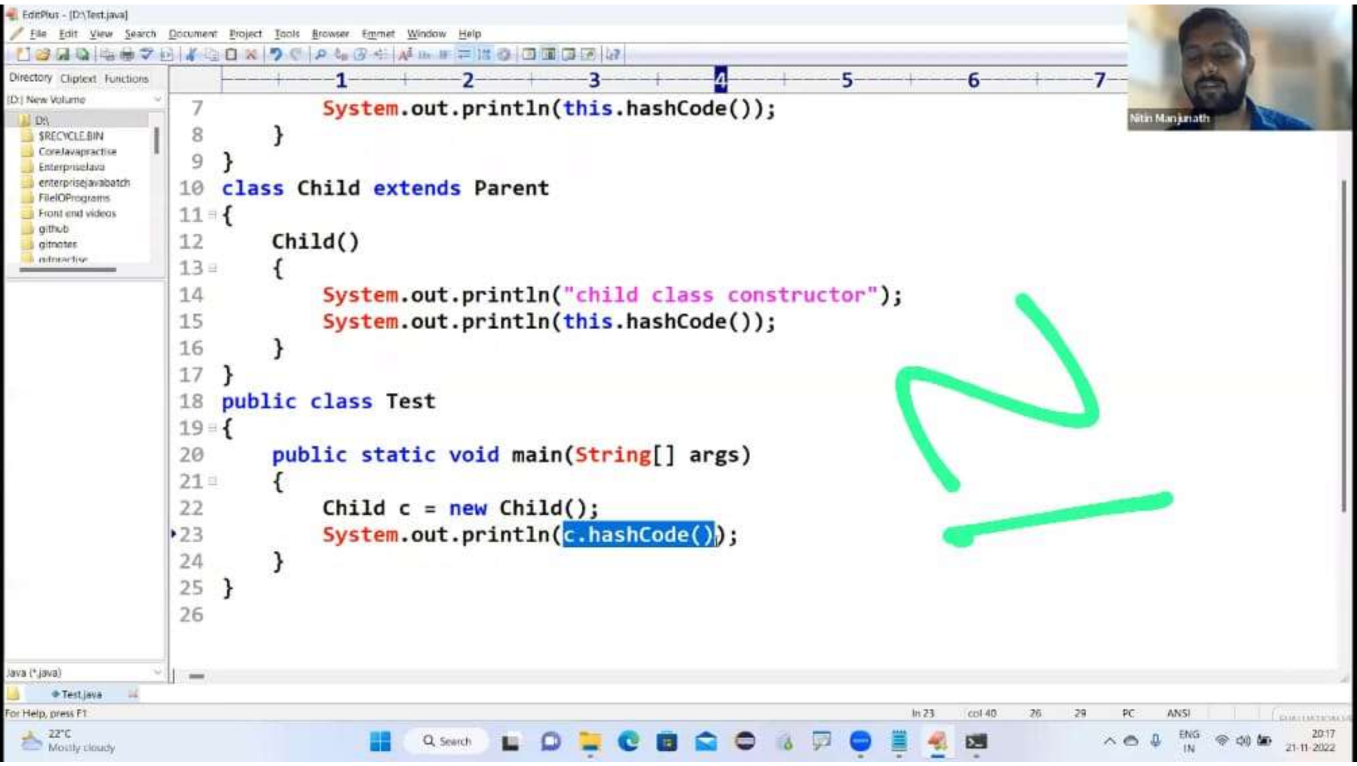
Interface:: Inside interface we can't write constructor.



```
              following modifiers like private,protected,transient,volatile.
 Abstract::  No restriction on access modifiers

 Interface:: For every interface variable compulsorily we should perform initialisation at the time of declaration,
              otherwise we get compile time error.
 Abstract::  Not required to perform initialisation for abstract class variables at the time of declaration.

 Interface:: Inside interface we can't write static and instance block.
 Abstract :: Inside abstract class we can write static and instance block.

 Interface:: Inside interface we can't write constructor.
 Abstract :: Inside abstract class we can write constructor.

Note:
static block     => .class file loading happends and used to initialize static variables.
instance block => during the creation of an object,just before the constructor call
used for initialization instance variable.
 constructor   => during the creation of an object, used for initialization instance variable.
```

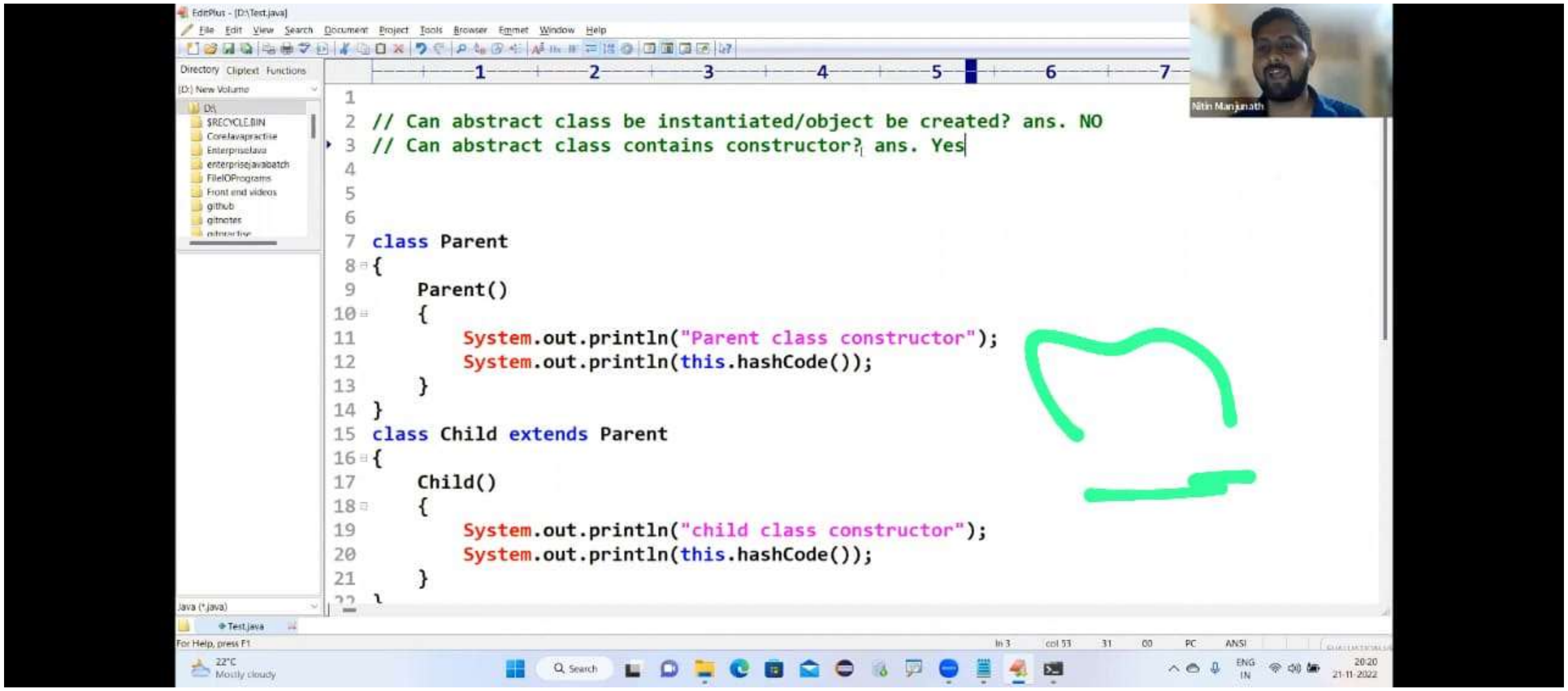


```java
//During the child class object creation, only Child class Object will be crated
// but no the parent class object(still constructor of parent is called to bring the
// properties of parent to child)
class Parent
{
    Parent()
    {
        System.out.println("Parent class constructor");
    }
}
class Child extends Parent
{
    Child()
    {
        System.out.println("child class constructor");
    }
}
public class Test
{
    public static void main(String[] args)
```



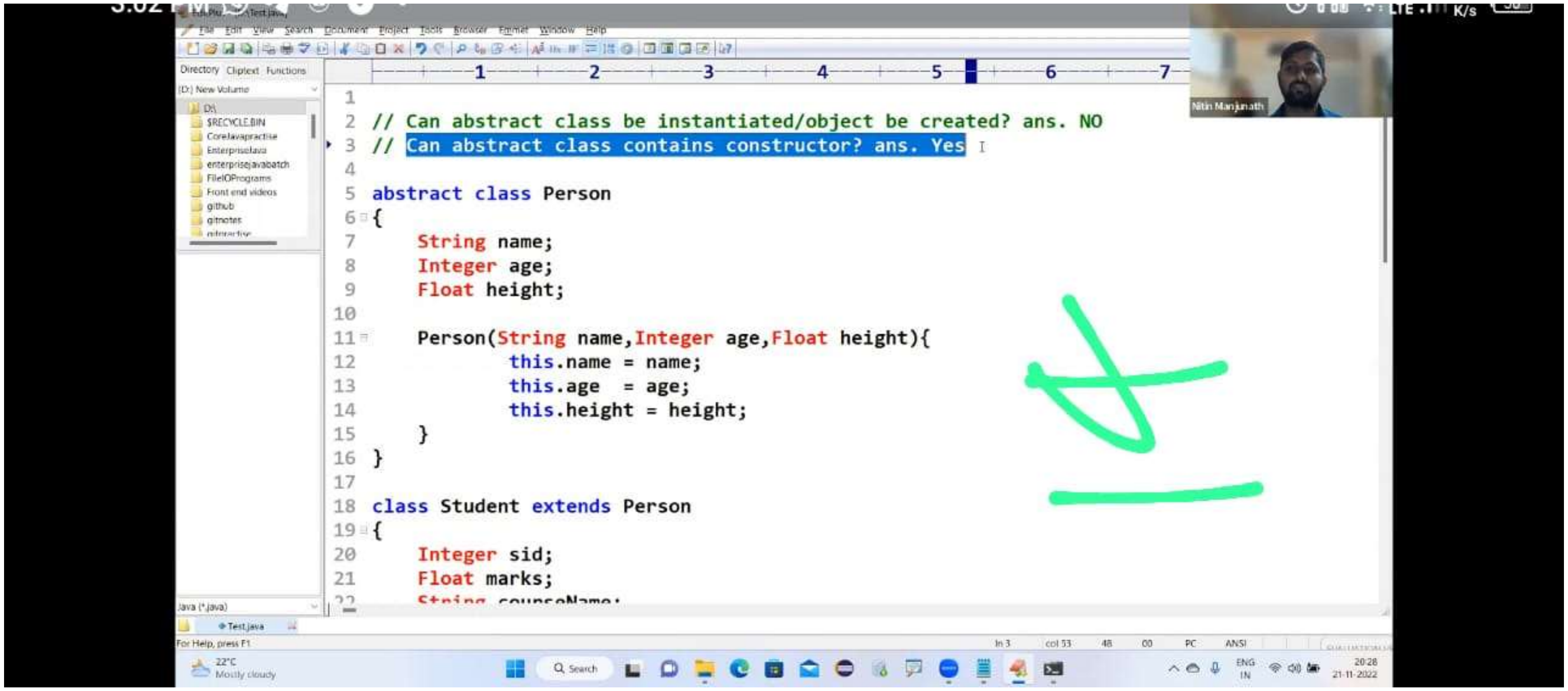
```
        System.out.println(this.hashCode());
    }
}
class Child extends Parent
{
    Child()
    {
        System.out.println("child class constructor");
        System.out.println(this.hashCode());
    }
}
public class Test
{
    public static void main(String[] args)
    {
        Child c = new Child();
        System.out.println(c.hashCode());
    }
}
```

12

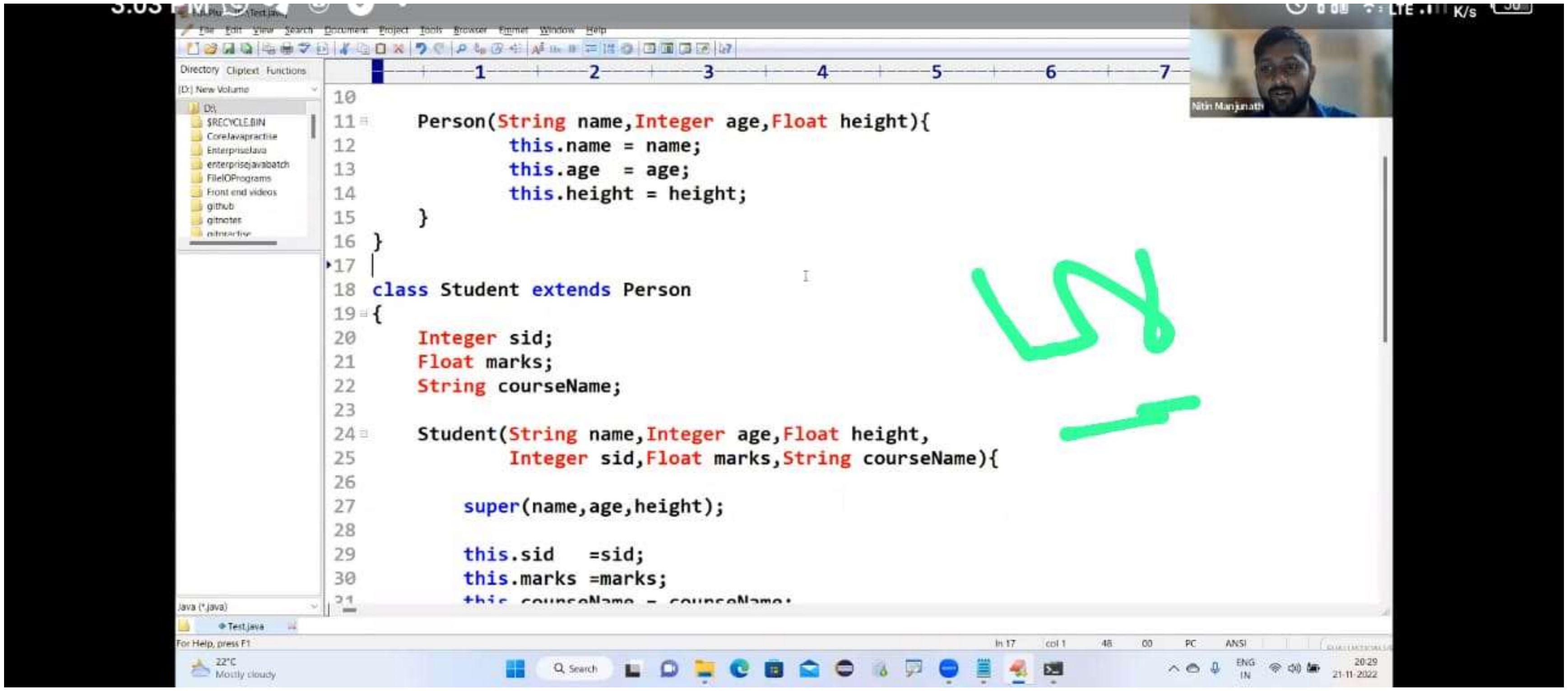

EditPlus - [D:\Test.java]
File Edit View Search Document Project Tools Browser Emmet Window Help
Nitin Manjunath

```java

// Can abstract class be instantiated/object be created? ans. NO
// Can abstract class contains constructor? ans. Yes



class Parent
{
    Parent()
    {
        System.out.println("Parent class constructor");
        System.out.println(this.hashCode());
    }
}
class Child extends Parent
{
    Child()
    {
        System.out.println("child class constructor");
        System.out.println(this.hashCode());
    }
```

13

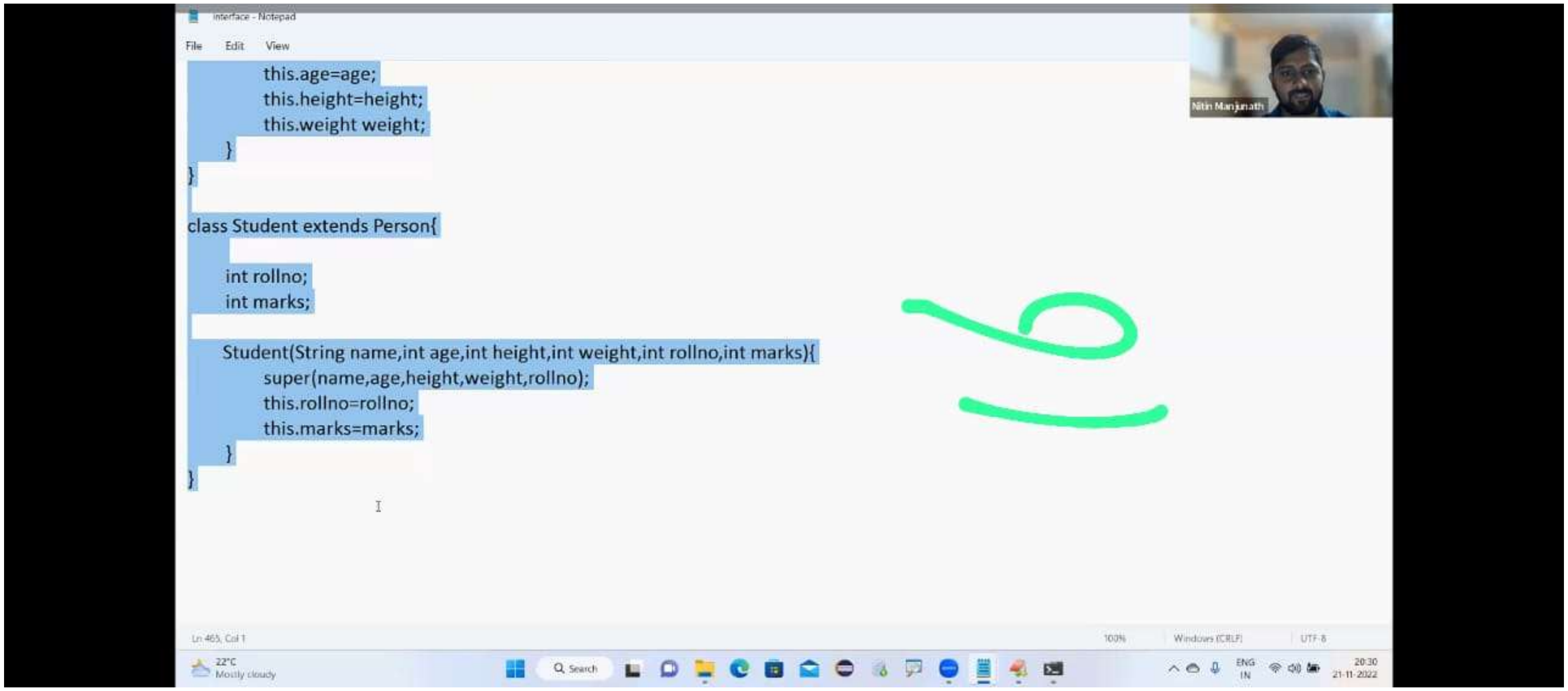


```java

// Can abstract class be instantiated/object be created? ans. NO
// Can abstract class contains constructor? ans. Yes

abstract class Person
{
    String name;
    Integer age;
    Float height;

    Person(String name,Integer age,Float height){
            this.name = name;
            this.age  = age;
            this.height = height;
    }
}

class Student extends Person
{
    Integer sid;
    Float marks;
```



```java
	Person(String name,Integer age,Float height){
		this.name = name;
		this.age  = age;
		this.height = height;
	}
}

class Student extends Person
{
	Integer sid;
	Float marks;
	String courseName;

	Student(String name,Integer age,Float height,
		Integer sid,Float marks,String courseName){

		super(name,age,height);

		this.sid   =sid;
		this.marks =marks;
		this.courseName = courseName;
```

15

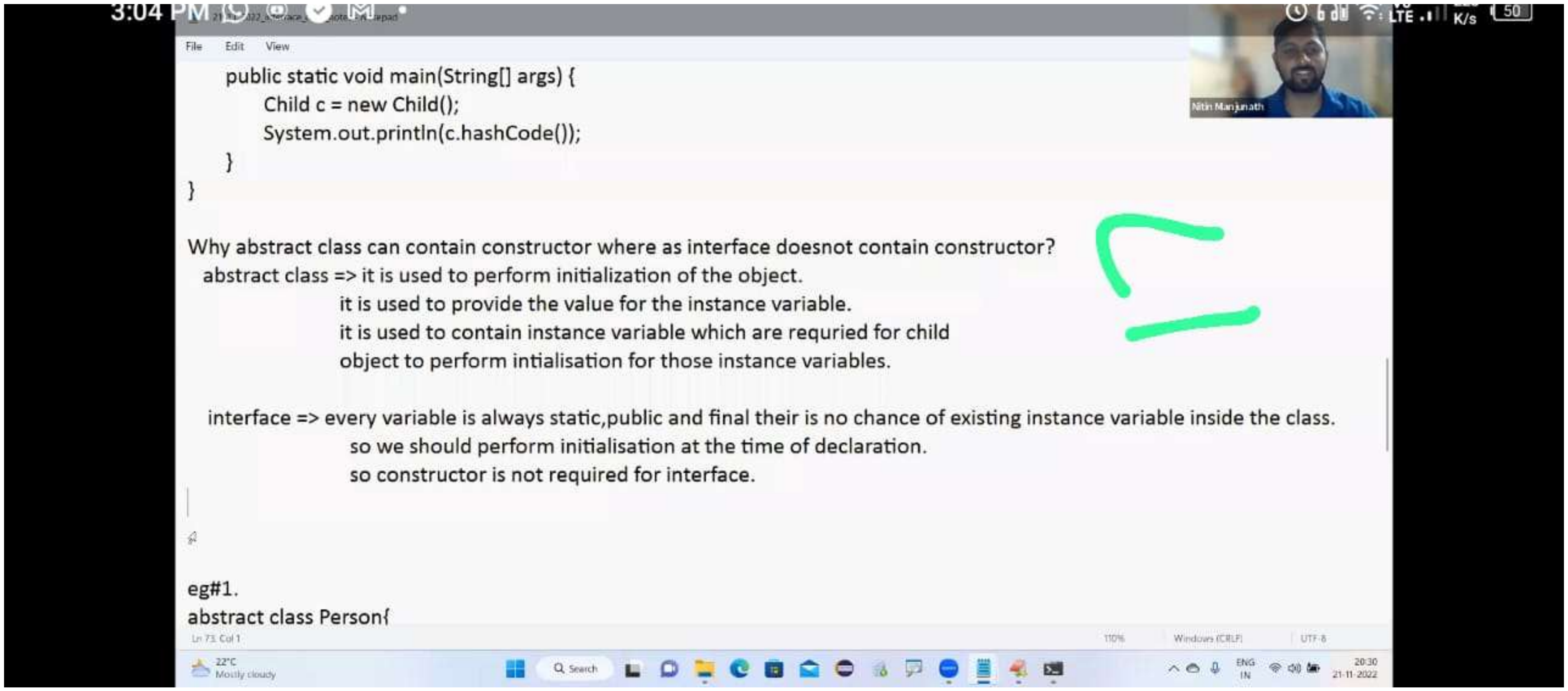


interface - Notepad

File Edit View

```
        this.age=age;
        this.height=height;
        this.weight weight;
    }
}

class Student extends Person{

    int rollno;
    int marks;

    Student(String name,int age,int height,int weight,int rollno,int marks){
        super(name,age,height,weight,rollno);
        this.rollno=rollno;
        this.marks=marks;
    }
}
```

Ln 465, Col 1 100% Windows (CRLF) UTF-8

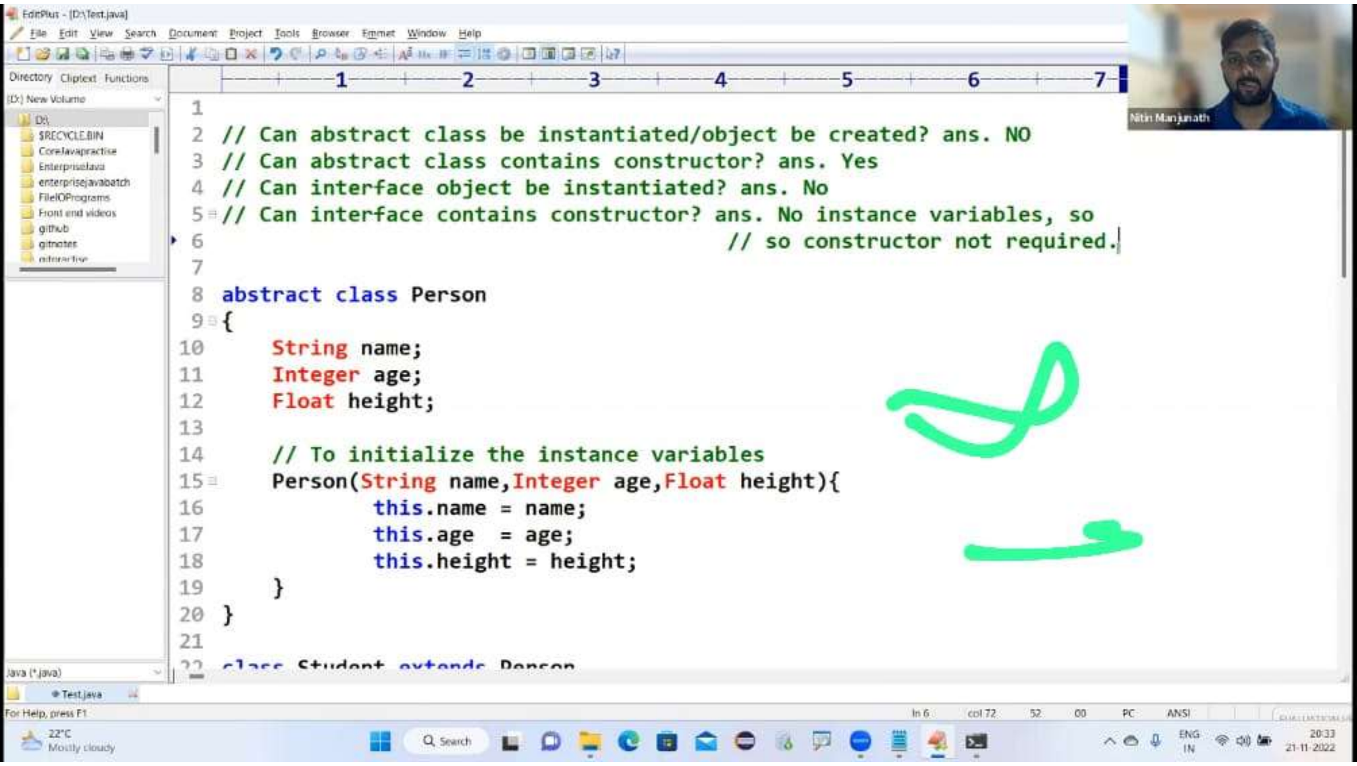


```
    public static void main(String[] args) {
        Child c = new Child();
        System.out.println(c.hashCode());
    }
}
```

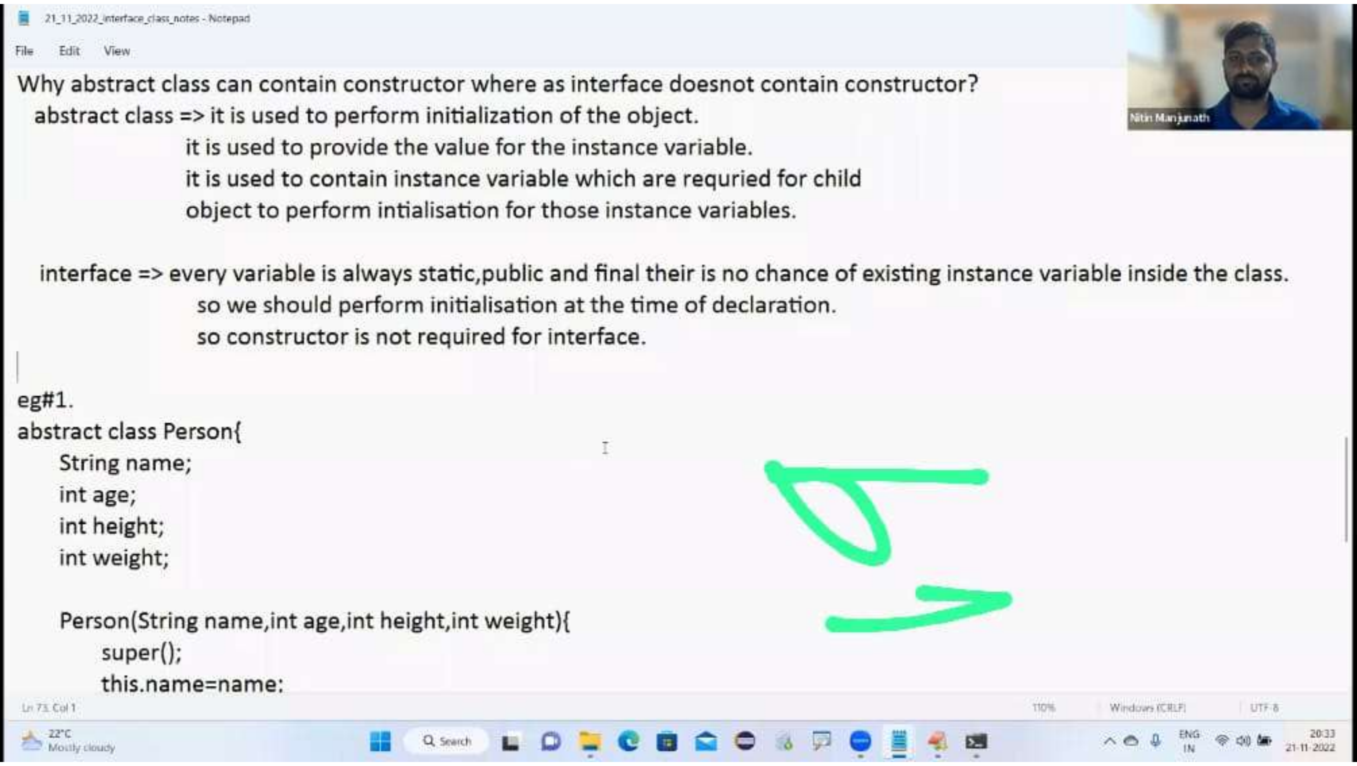
Why abstract class can contain constructor where as interface doesnot contain constructor?
  abstract class => it is used to perform initialization of the object.
                    it is used to provide the value for the instance variable.
                    it is used to contain instance variable which are requried for child
                    object to perform intialisation for those instance variables.

  interface => every variable is always static,public and final their is no chance of existing instance variable inside the class.
                    so we should perform initialisation at the time of declaration.
                    so constructor is not required for interface.

eg#1.
abstract class Person{



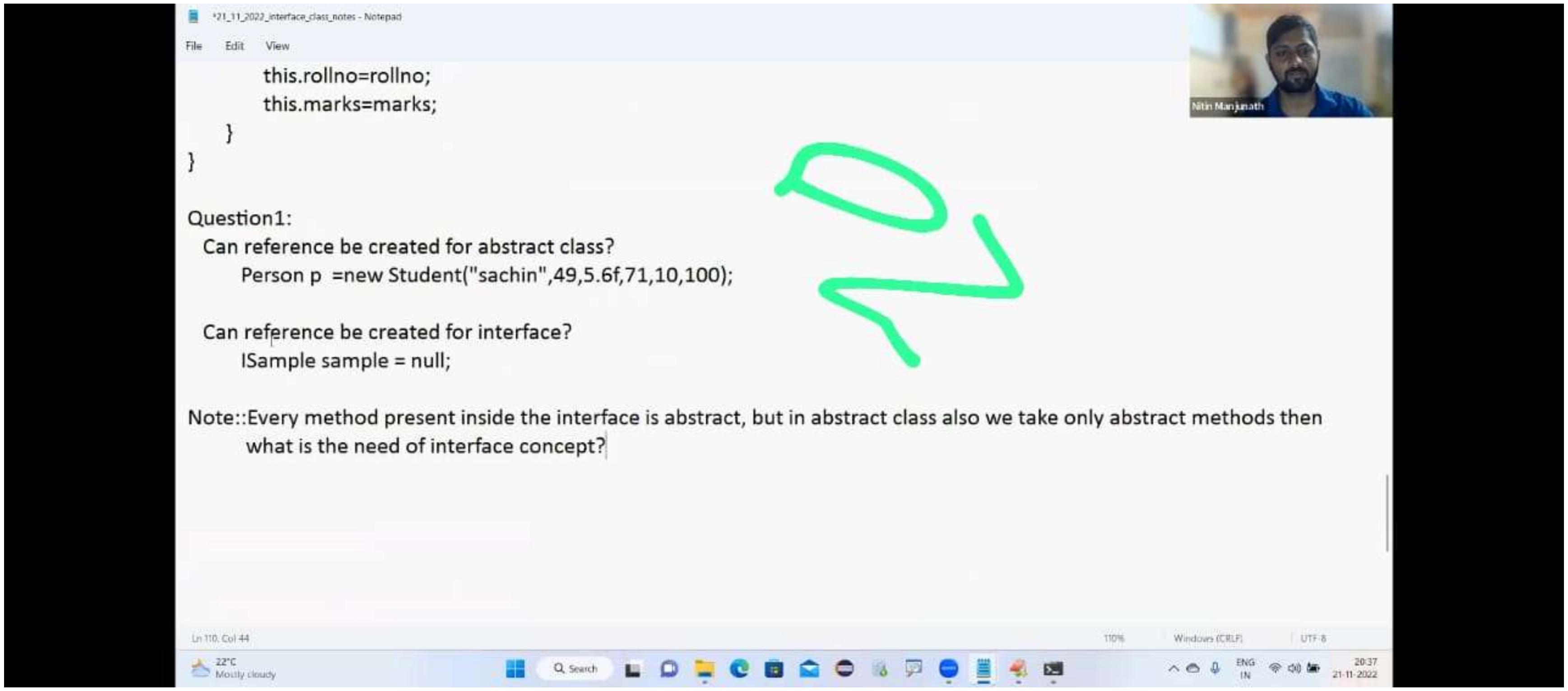
```java

// Can abstract class be instantiated/object be created? ans. NO
// Can abstract class contains constructor? ans. Yes
// Can interface object be instantiated? ans. No
// Can interface contains constructor? ans. No instance variables, so
                                   // so constructor not required.

abstract class Person
{
    String name;
    Integer age;
    Float height;

    // To initialize the instance variables
    Person(String name,Integer age,Float height){
            this.name = name;
            this.age  = age;
            this.height = height;
    }
}

class Student extends Person
```



Why abstract class can contain constructor where as interface doesnot contain constructor?
  abstract class => it is used to perform initialization of the object.
                 it is used to provide the value for the instance variable.
                 it is used to contain instance variable which are requried for child
                 object to perform intialisation for those instance variables.

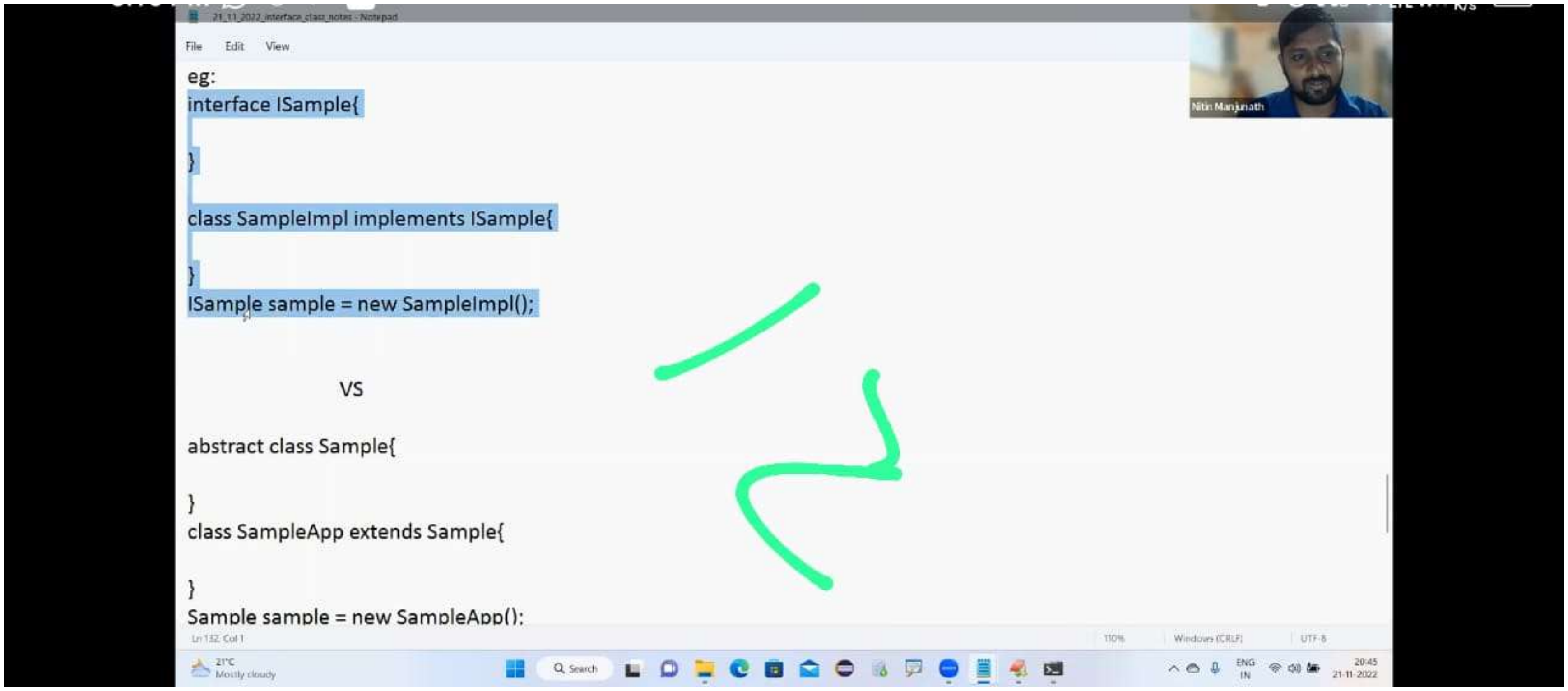
  interface => every variable is always static,public and final their is no chance of existing instance variable inside the class.
                 so we should perform initialisation at the time of declaration.
                 so constructor is not required for interface.

```
eg#1.
abstract class Person{
    String name;
    int age;
    int height;
    int weight;

    Person(String name,int age,int height,int weight){
        super();
        this.name=name;
```



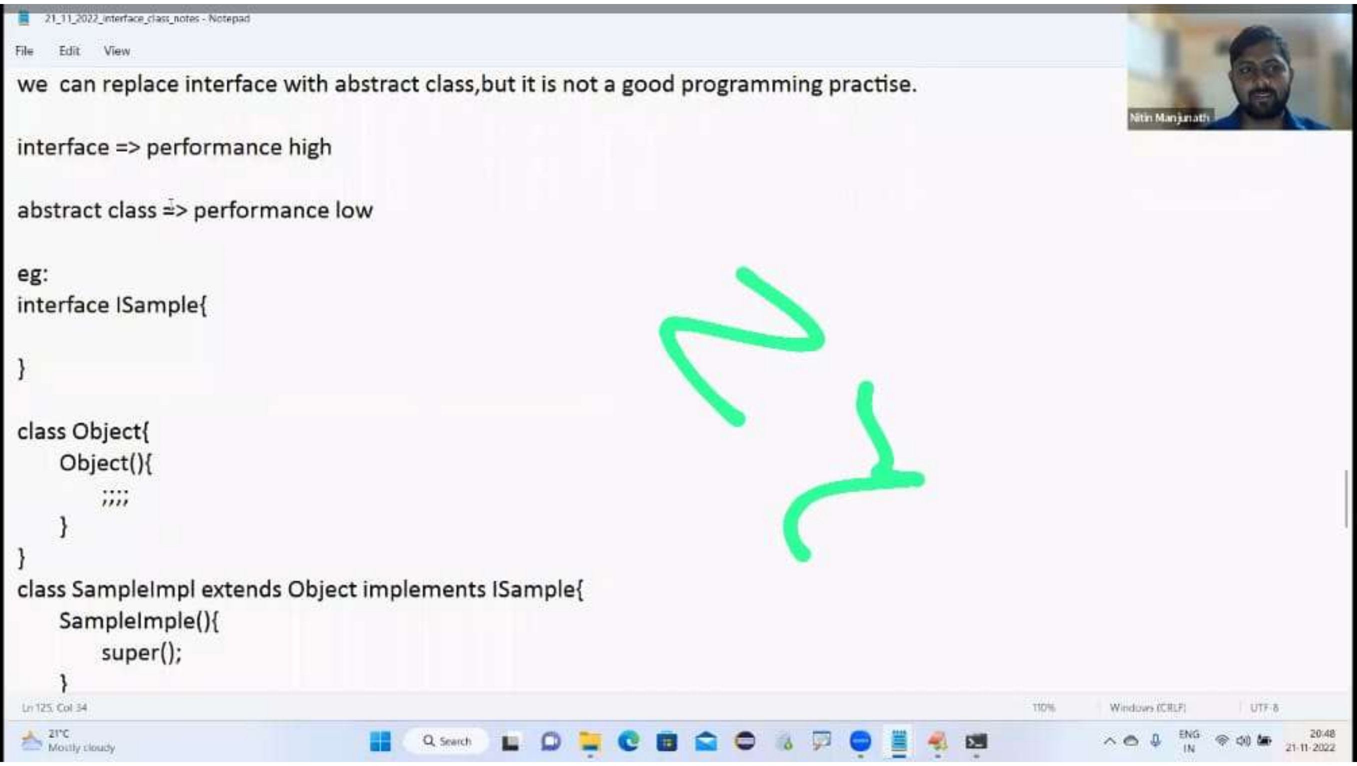
*21_11_2022_interface_class_notes - Notepad

```
        this.rollno=rollno;
        this.marks=marks;
    }
}

Question1:
  Can reference be created for abstract class?
      Person p  =new Student("sachin",49,5.6f,71,10,100);

  Can reference be created for interface?
      ISample sample = null;

Note::Every method present inside the interface is abstract, but in abstract class also we take only abstract methods then
      what is the need of interface concept?
```



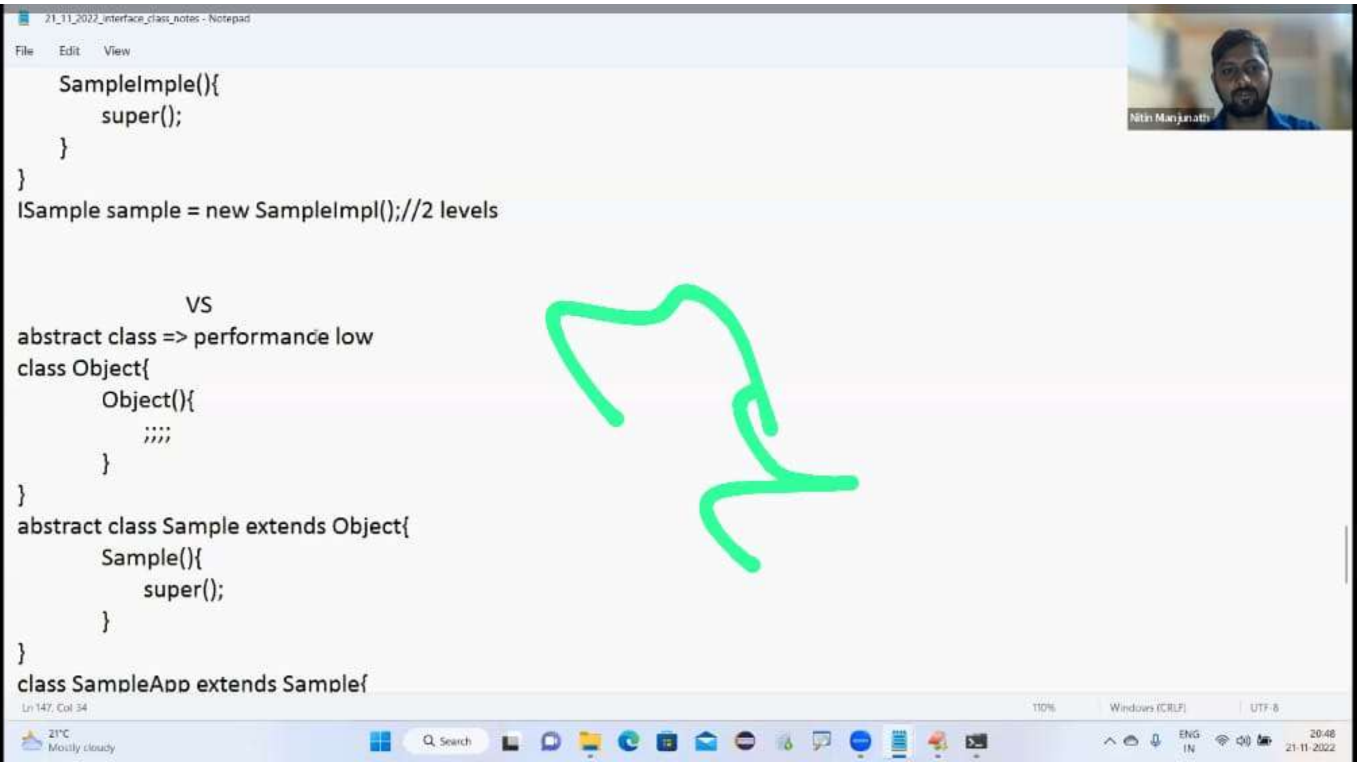
21_11_2022_interface_class_notes - Notepad

```
eg:
interface ISample{

}

class SampleImpl implements ISample{

}
ISample sample = new SampleImpl();


                VS

abstract class Sample{

}
class SampleApp extends Sample{

}
Sample sample = new SampleApp();
```



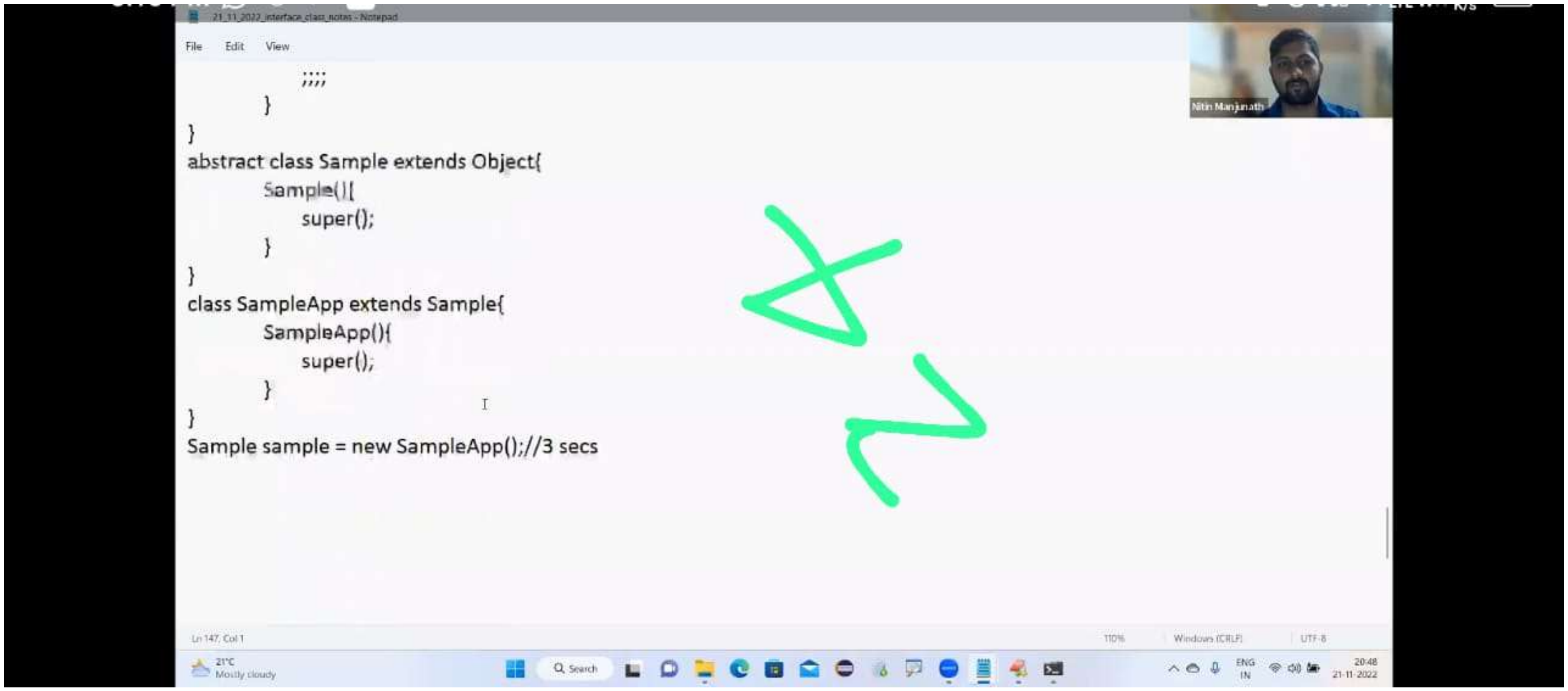
21_11_2022_interface_class_notes - Notepad

```
we  can replace interface with abstract class,but it is not a good programming practise.

interface => performance high

abstract class => performance low

eg:
interface ISample{

}

class Object{
    Object(){
        ;;;;
    }
}
class SampleImpl extends Object implements ISample{
    SampleImple(){
        super();
    }
```

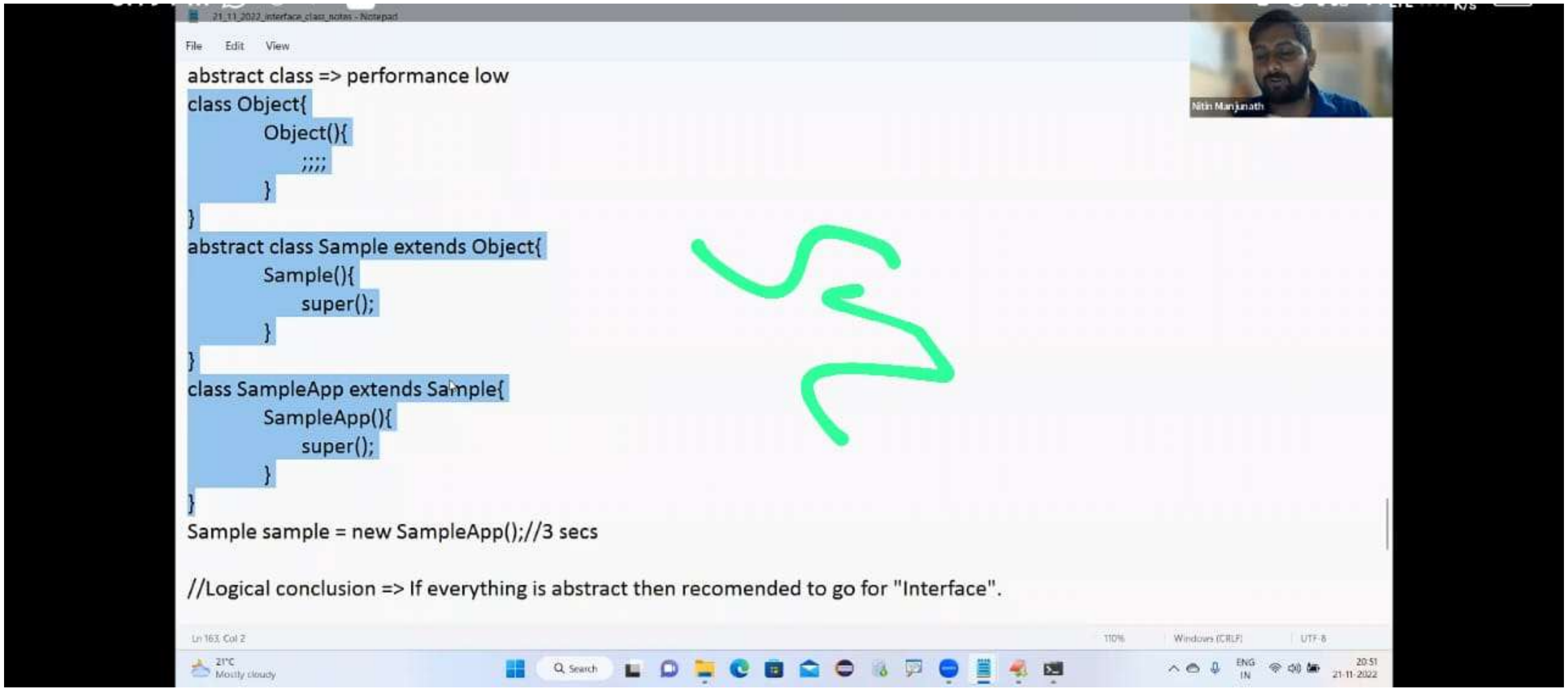


21_11_2022_interface_class_notes - Notepad

```
    SampleImple(){
        super();
    }
}
ISample sample = new SampleImpl();//2 levels



                VS
abstract class => performance low
class Object{
        Object(){
            ;;;;
        }
}
abstract class Sample extends Object{
        Sample(){
            super();
        }
}
class SampleApp extends Sample{
```

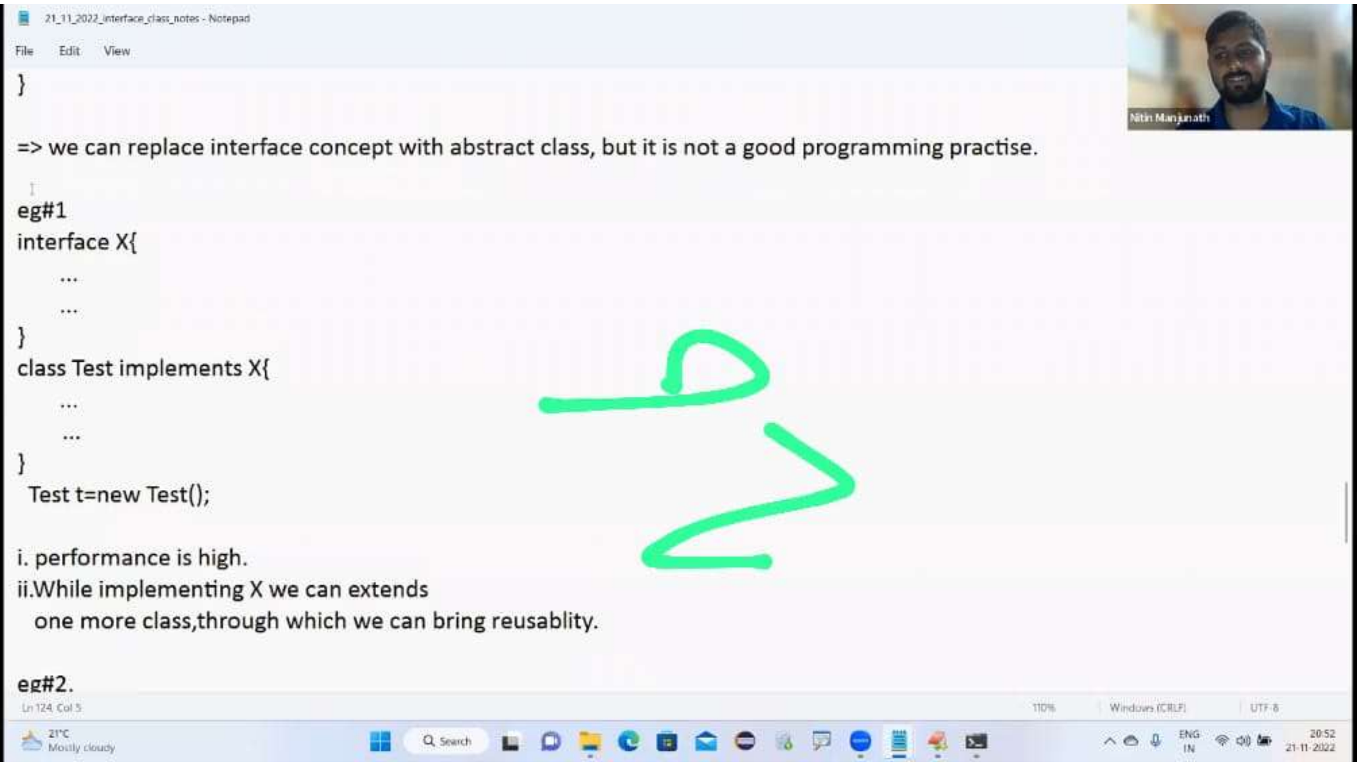


```
        ;;;;
    }
}
abstract class Sample extends Object{
    Sample(){
        super();
    }
}
class SampleApp extends Sample{
    SampleApp(){
        super();
    }
}
Sample sample = new SampleApp();//3 secs
```

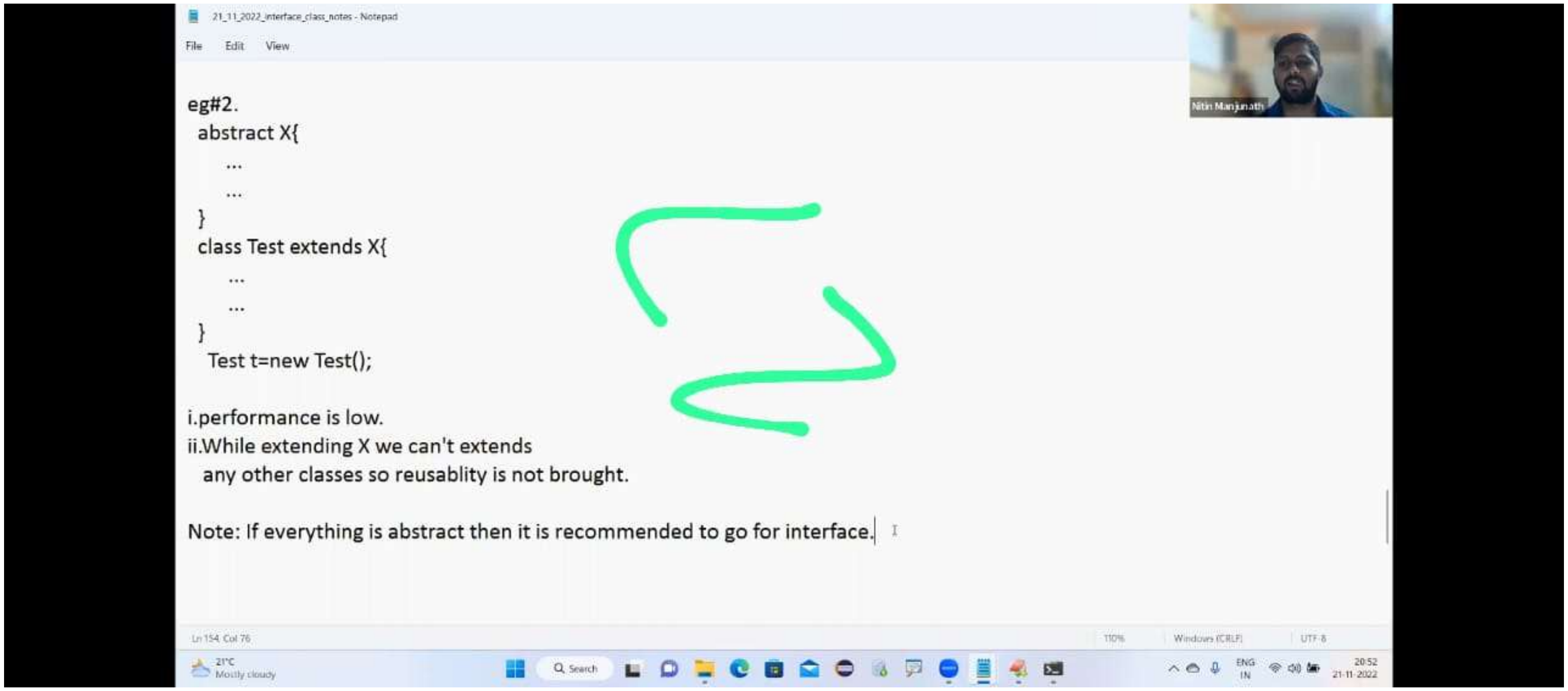


abstract class => performance low

```
class Object{
        Object(){
            ;;;;
        }
}
abstract class Sample extends Object{
        Sample(){
            super();
        }
}
class SampleApp extends Sample{
        SampleApp(){
            super();
        }
}
Sample sample = new SampleApp();//3 secs

//Logical conclusion => If everything is abstract then recomended to go for "Interface".
```



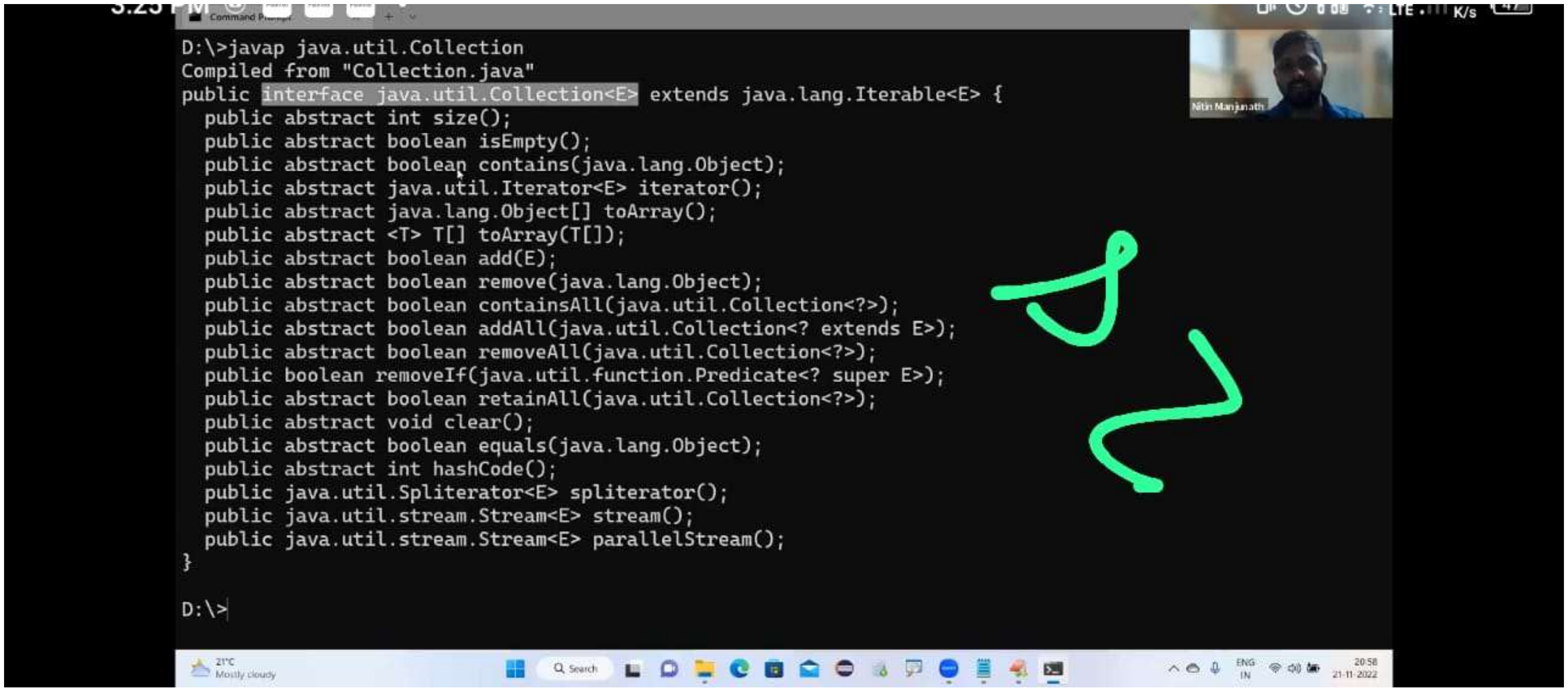
```
}

=> we can replace interface concept with abstract class, but it is not a good programming practise.

eg#1
interface X{
      ...
      ...
}
class Test implements X{
      ...
      ...
}
 Test t=new Test();

i. performance is high.
ii.While implementing X we can extends
  one more class,through which we can bring reusablity.

eg#2.
```



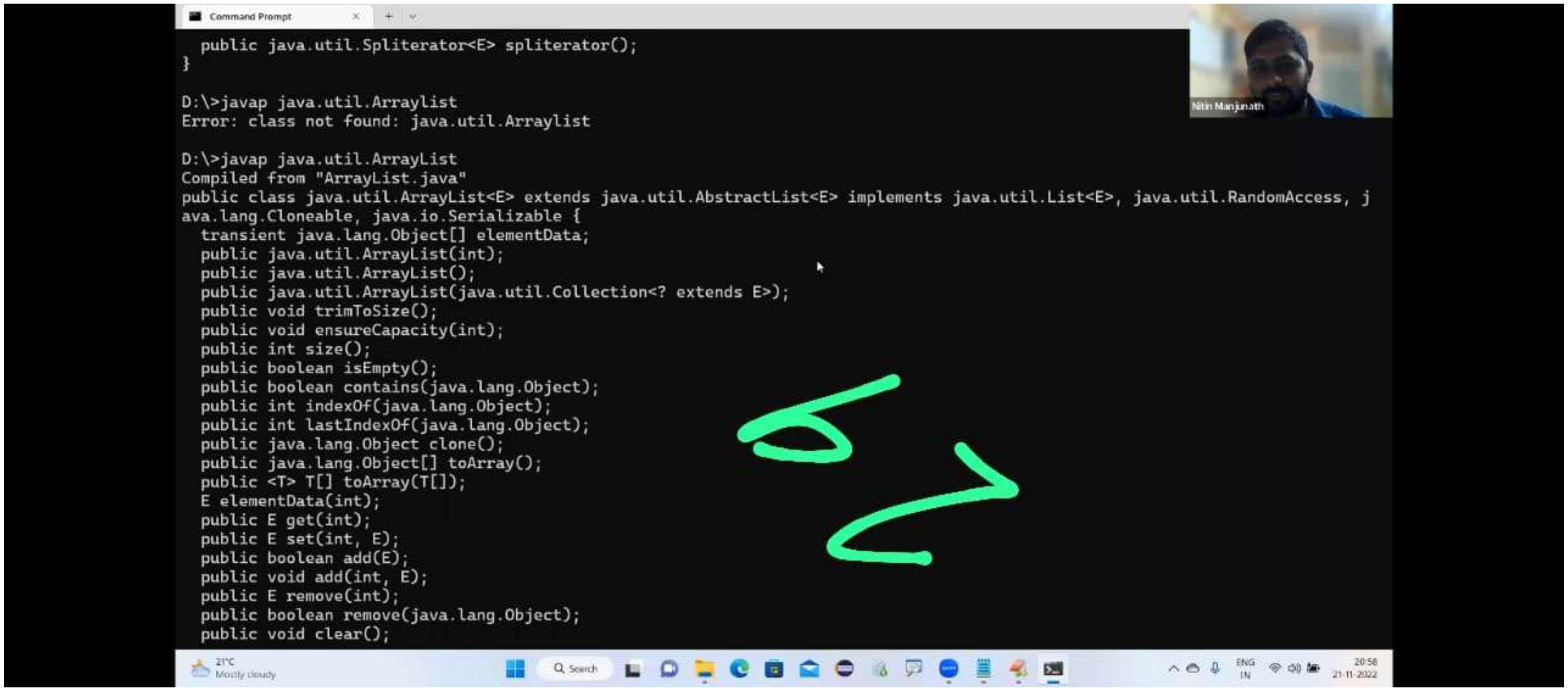
```
eg#2.
 abstract X{
     ...
     ...
 }
 class Test extends X{
     ...
     ...
 }
  Test t=new Test();
```

i.performance is low.
ii.While extending X we can't extends
  any other classes so reusablity is not brought.

Note: If everything is abstract then it is recommended to go for interface.

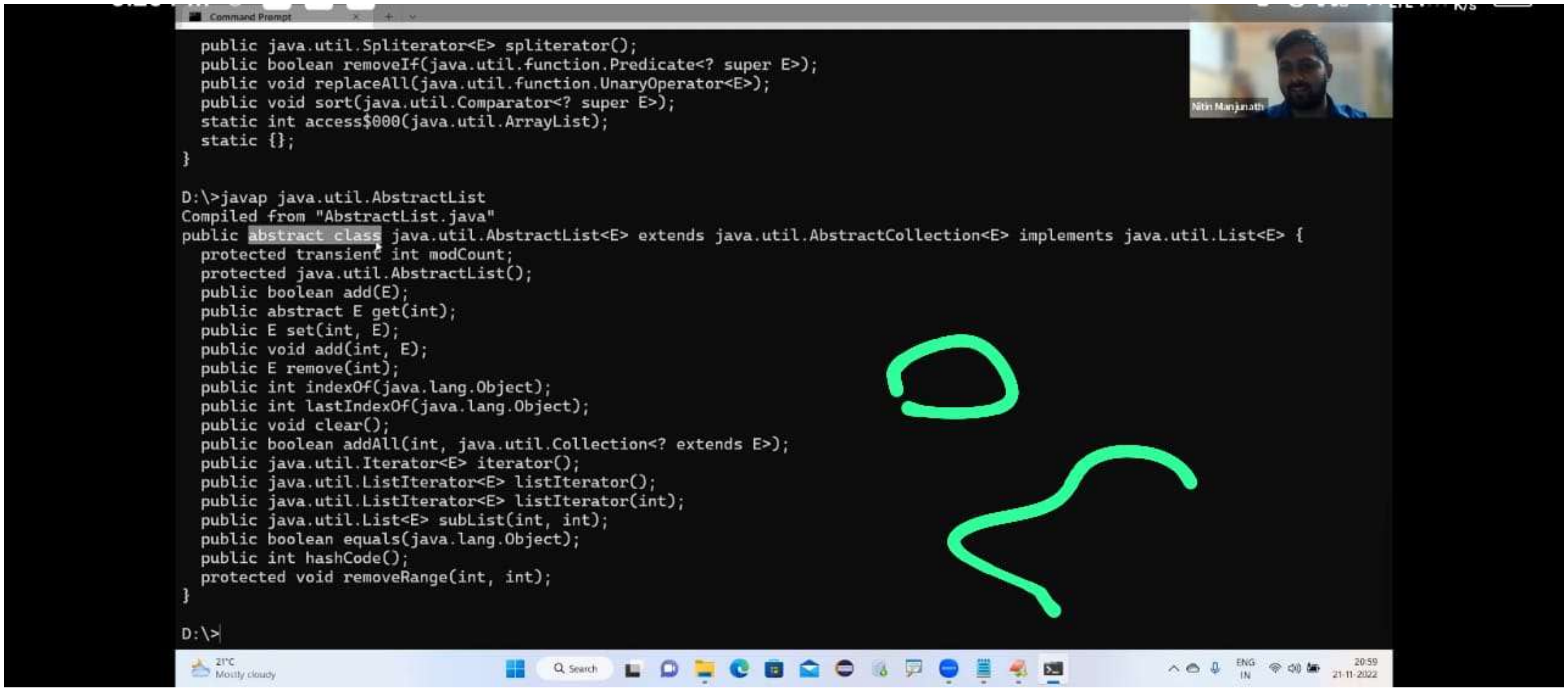


```
D:\>javap java.util.Collection
Compiled from "Collection.java"
public interface java.util.Collection<E> extends java.lang.Iterable<E> {
  public abstract int size();
  public abstract boolean isEmpty();
  public abstract boolean contains(java.lang.Object);
  public abstract java.util.Iterator<E> iterator();
  public abstract java.lang.Object[] toArray();
  public abstract <T> T[] toArray(T[]);
  public abstract boolean add(E);
  public abstract boolean remove(java.lang.Object);
  public abstract boolean containsAll(java.util.Collection<?>);
  public abstract boolean addAll(java.util.Collection<? extends E>);
  public abstract boolean removeAll(java.util.Collection<?>);
  public boolean removeIf(java.util.function.Predicate<? super E>);
  public abstract boolean retainAll(java.util.Collection<?>);
  public abstract void clear();
  public abstract boolean equals(java.lang.Object);
  public abstract int hashCode();
  public java.util.Spliterator<E> spliterator();
  public java.util.stream.Stream<E> stream();
  public java.util.stream.Stream<E> parallelStream();
}

D:\>
```



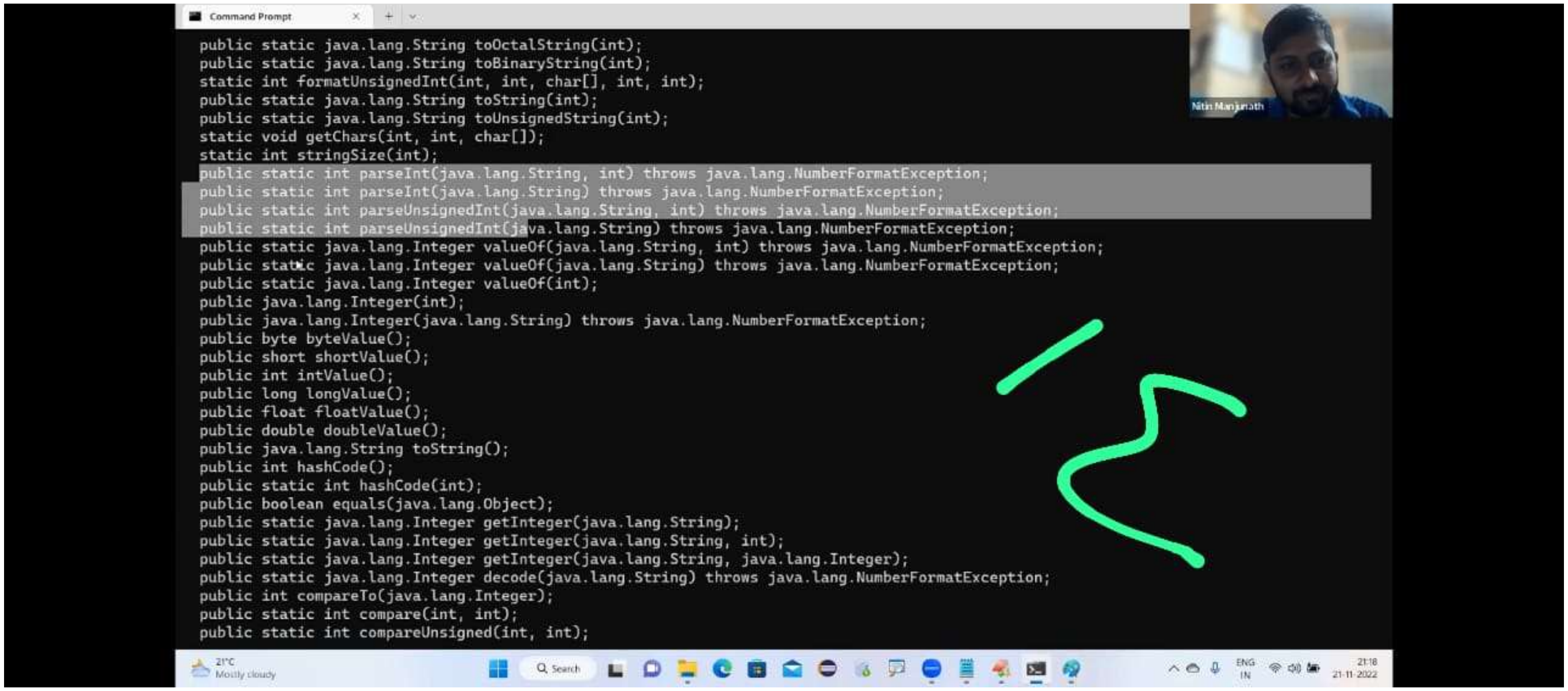
```
  public java.util.Spliterator<E> spliterator();
}

D:\>javap java.util.Arraylist
Error: class not found: java.util.Arraylist

D:\>javap java.util.ArrayList
Compiled from "ArrayList.java"
public class java.util.ArrayList<E> extends java.util.AbstractList<E> implements java.util.List<E>, java.util.RandomAccess, j
ava.lang.Cloneable, java.io.Serializable {
  transient java.lang.Object[] elementData;
  public java.util.ArrayList(int);
  public java.util.ArrayList();
  public java.util.ArrayList(java.util.Collection<? extends E>);
  public void trimToSize();
  public void ensureCapacity(int);
  public int size();
  public boolean isEmpty();
  public boolean contains(java.lang.Object);
  public int indexOf(java.lang.Object);
  public int lastIndexOf(java.lang.Object);
  public java.lang.Object clone();
  public java.lang.Object[] toArray();
  public <T> T[] toArray(T[]);
  E elementData(int);
  public E get(int);
  public E set(int, E);
  public boolean add(E);
  public void add(int, E);
  public E remove(int);
  public boolean remove(java.lang.Object);
  public void clear();
```

29



```
  public java.util.Spliterator<E> spliterator();
  public boolean removeIf(java.util.function.Predicate<? super E>);
  public void replaceAll(java.util.function.UnaryOperator<E>);
  public void sort(java.util.Comparator<? super E>);
  static int access$000(java.util.ArrayList);
  static {};
}

D:\>javap java.util.AbstractList
Compiled from "AbstractList.java"
public abstract class java.util.AbstractList<E> extends java.util.AbstractCollection<E> implements java.util.List<E> {
  protected transient int modCount;
  protected java.util.AbstractList();
  public boolean add(E);
  public abstract E get(int);
  public E set(int, E);
  public void add(int, E);
  public E remove(int);
  public int indexOf(java.lang.Object);
  public int lastIndexOf(java.lang.Object);
  public void clear();
  public boolean addAll(int, java.util.Collection<? extends E>);
  public java.util.Iterator<E> iterator();
  public java.util.ListIterator<E> listIterator();
  public java.util.ListIterator<E> listIterator(int);
  public java.util.List<E> subList(int, int);
  public boolean equals(java.lang.Object);
  public int hashCode();
  protected void removeRange(int, int);
}

D:\>
```

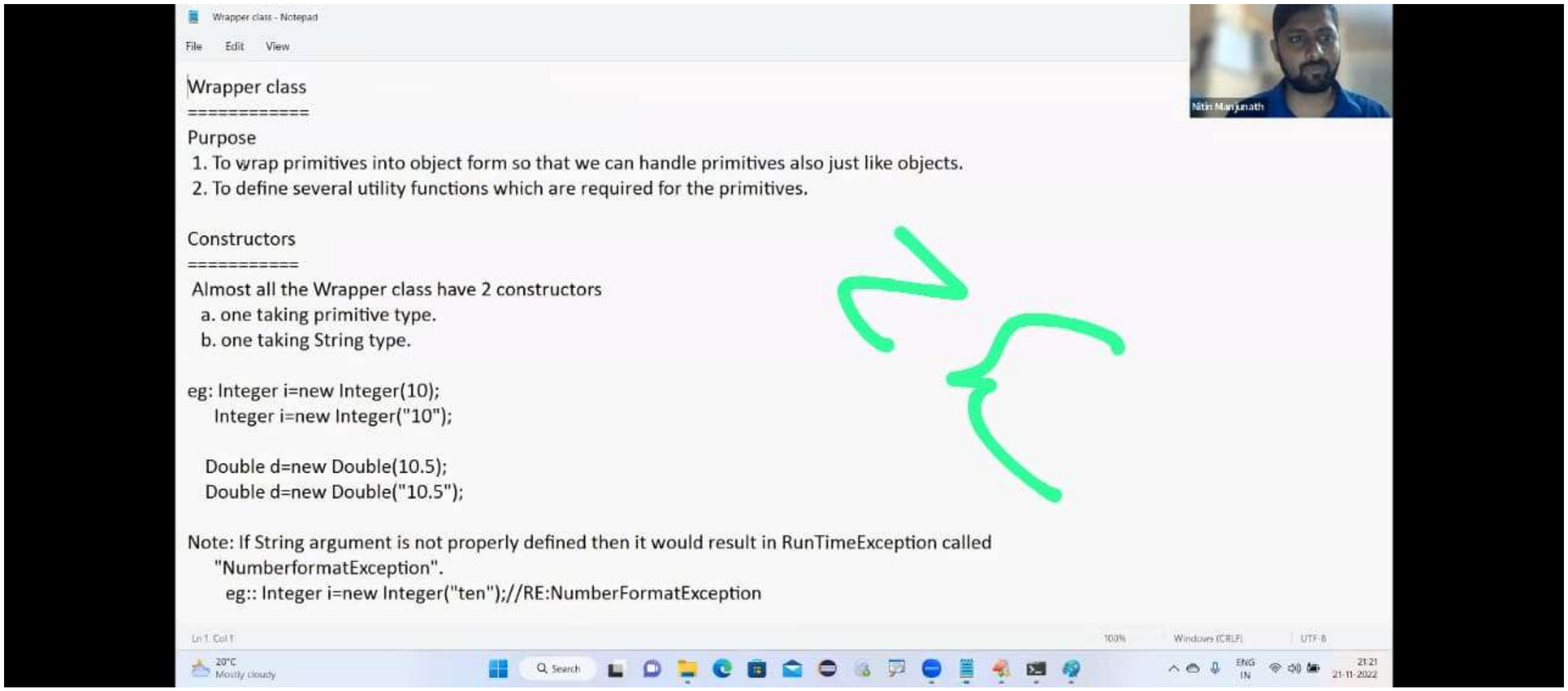
30



Command Prompt

```
public static java.lang.String toOctalString(int);
public static java.lang.String toBinaryString(int);
static int formatUnsignedInt(int, int, char[], int, int);
public static java.lang.String toString(int);
public static java.lang.String toUnsignedString(int);
static void getChars(int, int, char[]);
static int stringSize(int);
public static int parseInt(java.lang.String, int) throws java.lang.NumberFormatException;
public static int parseInt(java.lang.String) throws java.lang.NumberFormatException;
public static int parseUnsignedInt(java.lang.String, int) throws java.lang.NumberFormatException;
public static int parseUnsignedInt(java.lang.String) throws java.lang.NumberFormatException;
public static java.lang.Integer valueOf(java.lang.String, int) throws java.lang.NumberFormatException;
public static java.lang.Integer valueOf(java.lang.String) throws java.lang.NumberFormatException;
public static java.lang.Integer valueOf(int);
public java.lang.Integer(int);
public java.lang.Integer(java.lang.String) throws java.lang.NumberFormatException;
public byte byteValue();
public short shortValue();
public int intValue();
public long longValue();
public float floatValue();
public double doubleValue();
public java.lang.String toString();
public int hashCode();
public static int hashCode(int);
public boolean equals(java.lang.Object);
public static java.lang.Integer getInteger(java.lang.String);
public static java.lang.Integer getInteger(java.lang.String, int);
public static java.lang.Integer getInteger(java.lang.String, java.lang.Integer);
public static java.lang.Integer decode(java.lang.String) throws java.lang.NumberFormatException;
public int compareTo(java.lang.Integer);
public static int compare(int, int);
public static int compareUnsigned(int, int);
```

31



Wrapper class
============
Purpose
1. To wrap primitives into object form so that we can handle primitives also just like objects.
2. To define several utility functions which are required for the primitives.

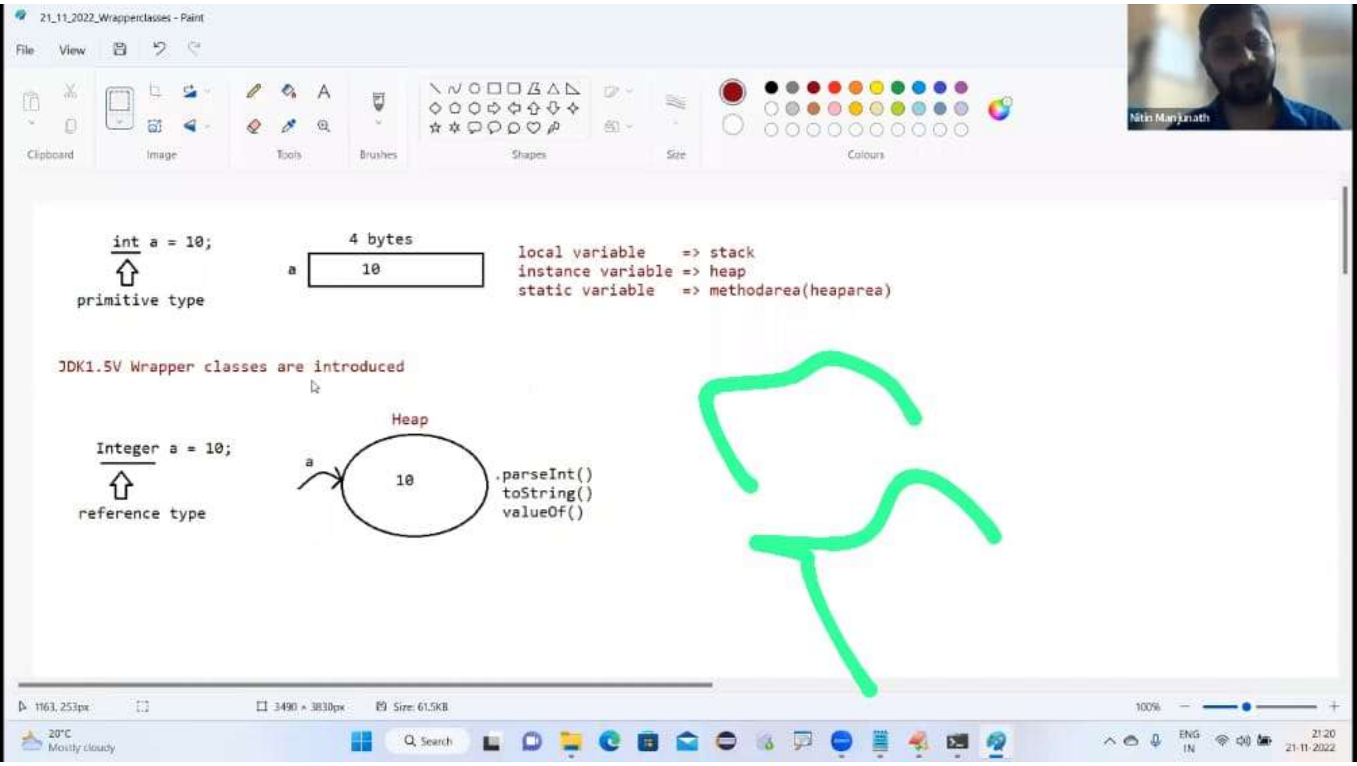
Constructors
===========
Almost all the Wrapper class have 2 constructors
a. one taking primitive type.
b. one taking String type.

```
eg: Integer i=new Integer(10);
    Integer i=new Integer("10");

  Double d=new Double(10.5);
  Double d=new Double("10.5");
```

Note: If String argument is not properly defined then it would result in RunTimeException called "NumberformatException".

```
    eg:: Integer i=new Integer("ten");//RE:NumberFormatException
```

32



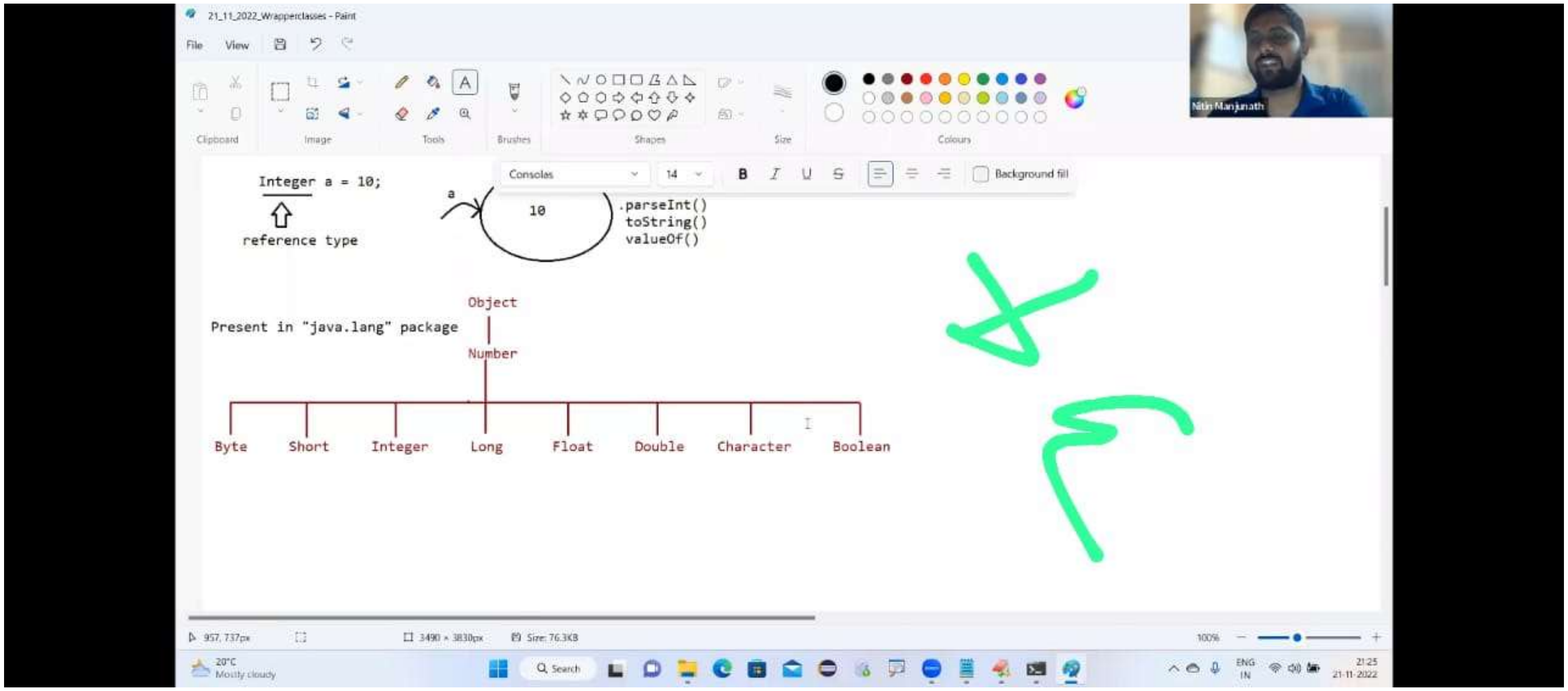
```
int a = 10;
primitive type
```

```
local variable    => stack
instance variable => heap
static variable   => methodarea(heaparea)
```

JDK1.5V Wrapper classes are introduced

```
Integer a = 10;
reference type
```

```
.parseInt()
 toString()
 valueOf()
```



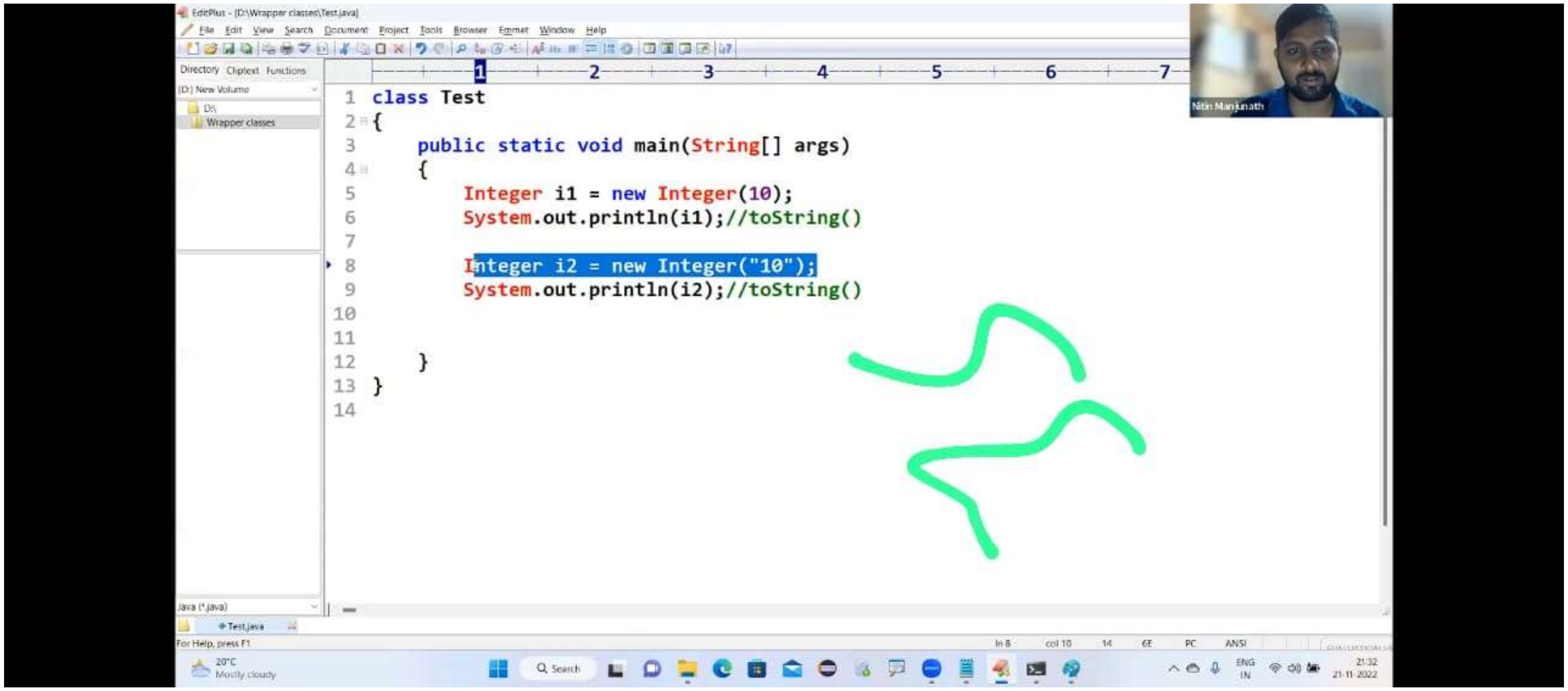
```
Integer a = 10;
⇧
reference type
```

a → 10

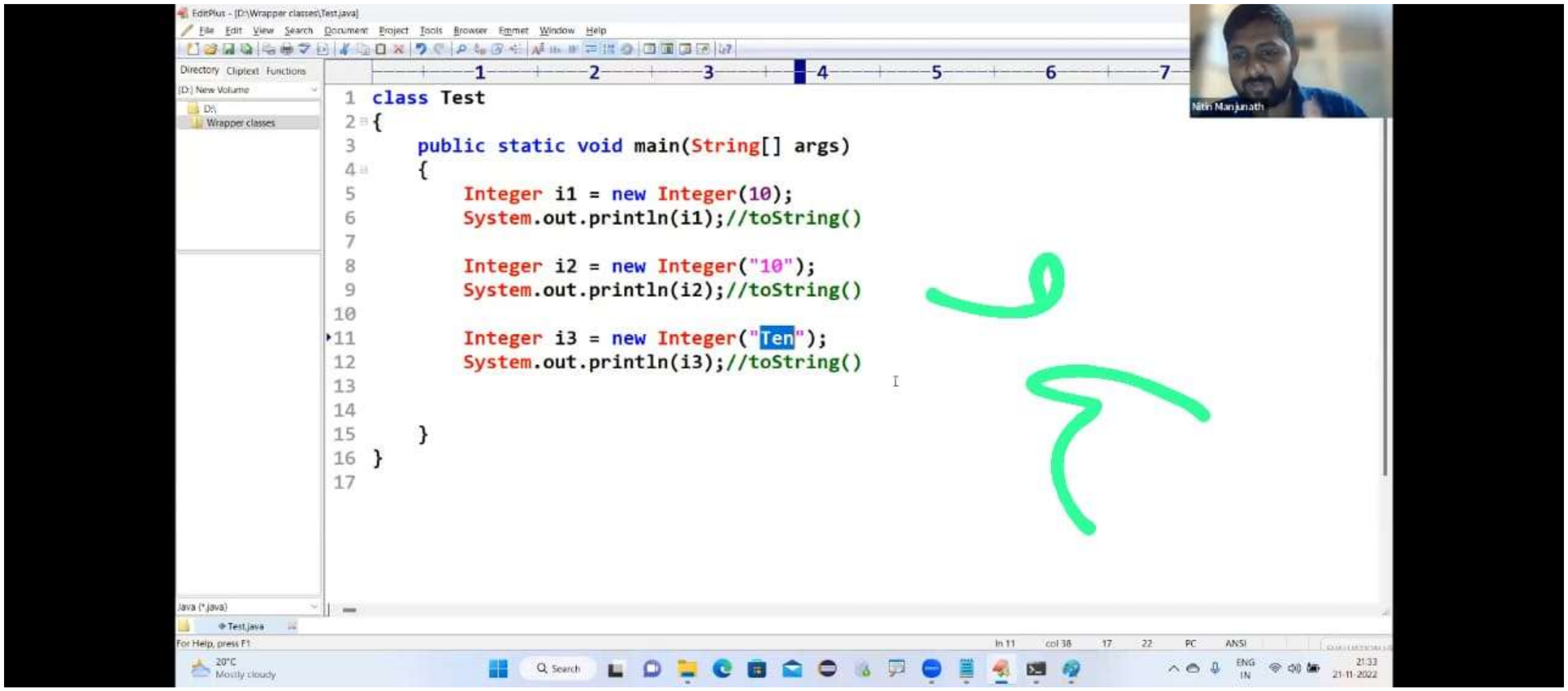
```
.parseInt()
toString()
valueOf()
```

Present in "java.lang" package

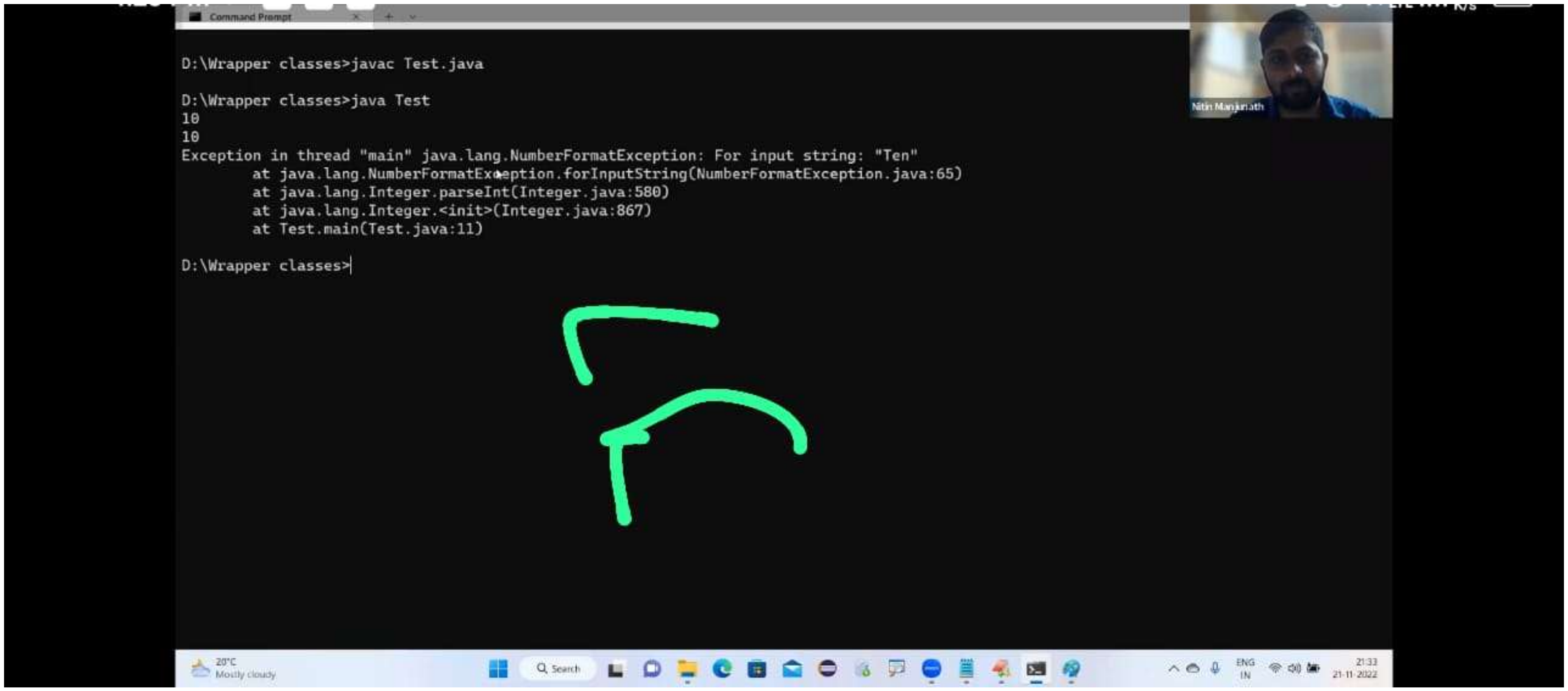
Object
Number
Byte Short Integer Long Float Double Character Boolean



```java
class Test
{
    public static void main(String[] args)
    {
        Integer i1 = new Integer(10);
        System.out.println(i1);//toString()

        Integer i2 = new Integer("10");
        System.out.println(i2);//toString()


    }
}

```

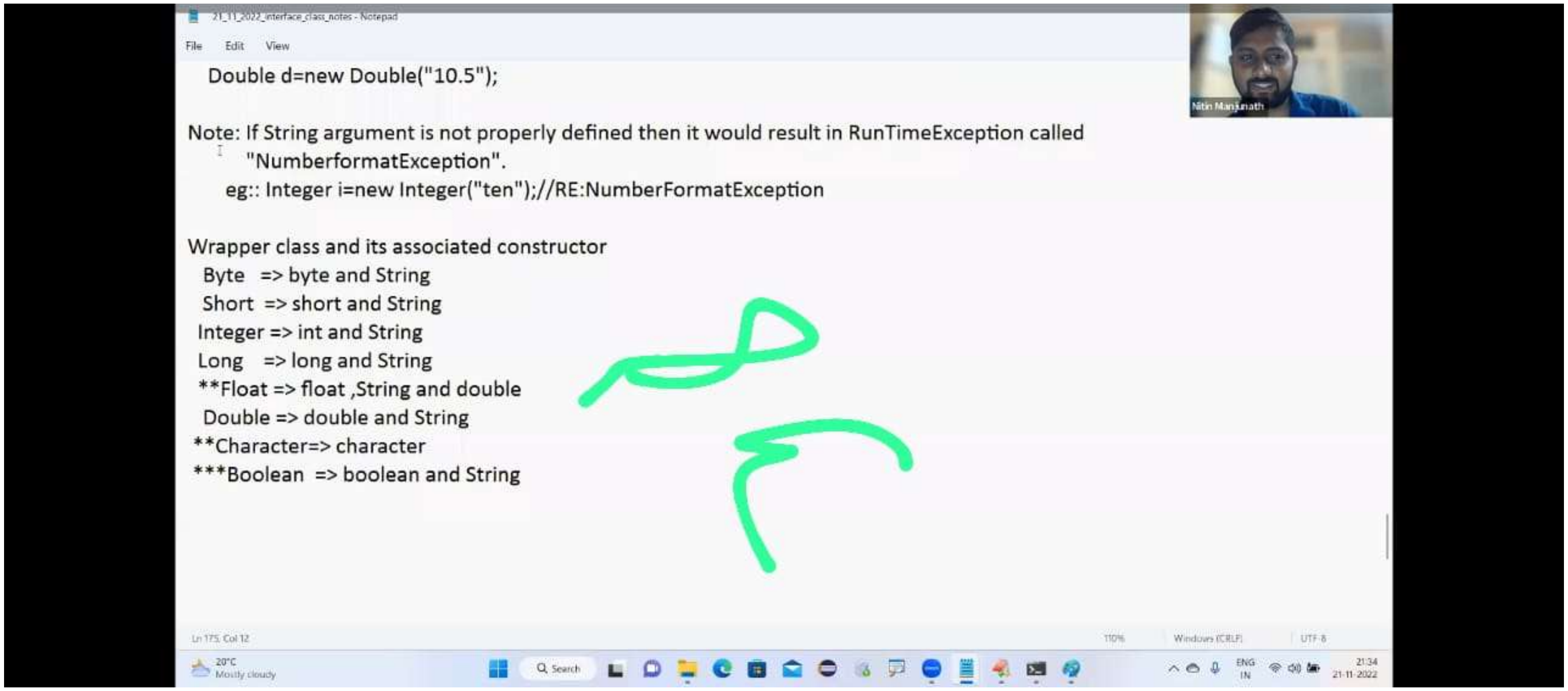


```java
class Test
{
    public static void main(String[] args)
    {
        Integer i1 = new Integer(10);
        System.out.println(i1);//toString()

        Integer i2 = new Integer("10");
        System.out.println(i2);//toString()

        Integer i3 = new Integer("Ten");
        System.out.println(i3);//toString()


    }
}
```

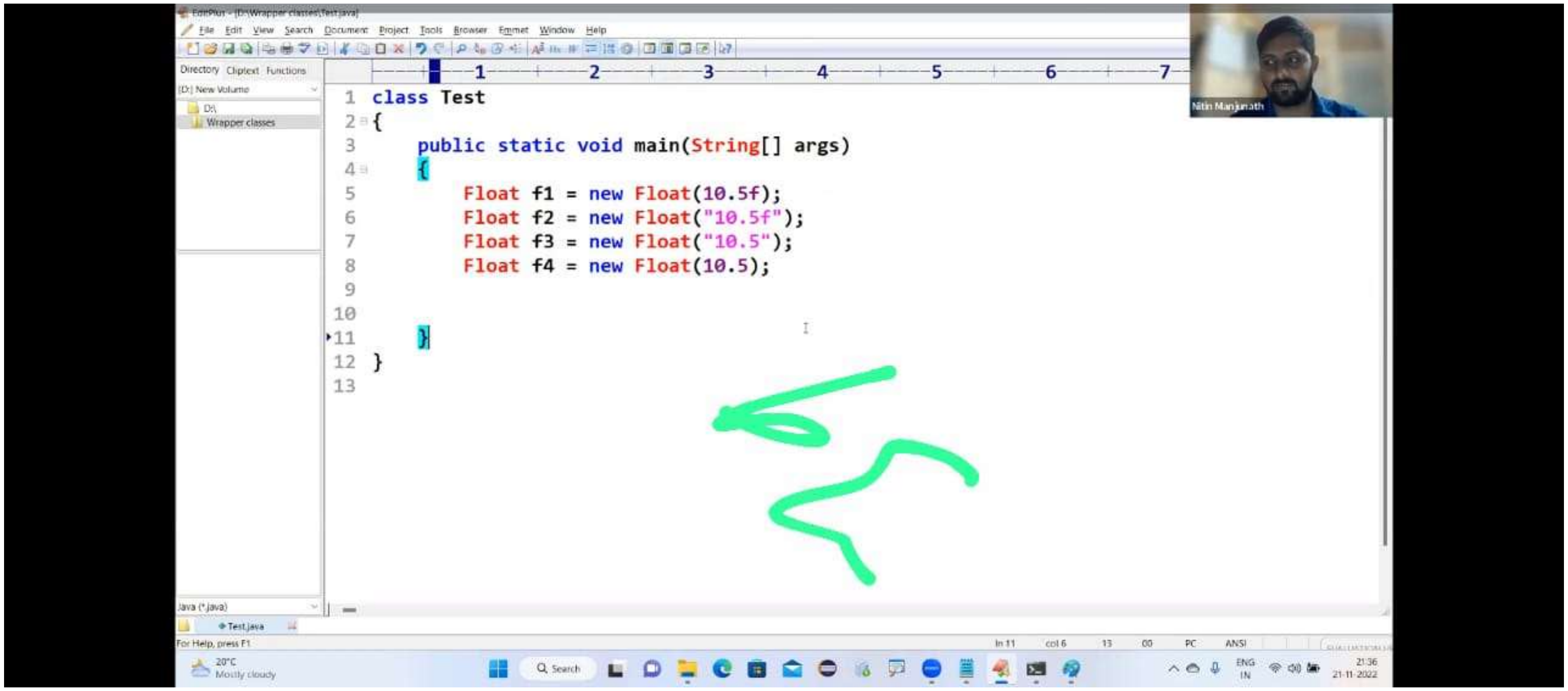


```
D:\Wrapper classes>javac Test.java

D:\Wrapper classes>java Test
10
10
Exception in thread "main" java.lang.NumberFormatException: For input string: "Ten"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.<init>(Integer.java:867)
        at Test.main(Test.java:11)

D:\Wrapper classes>
```

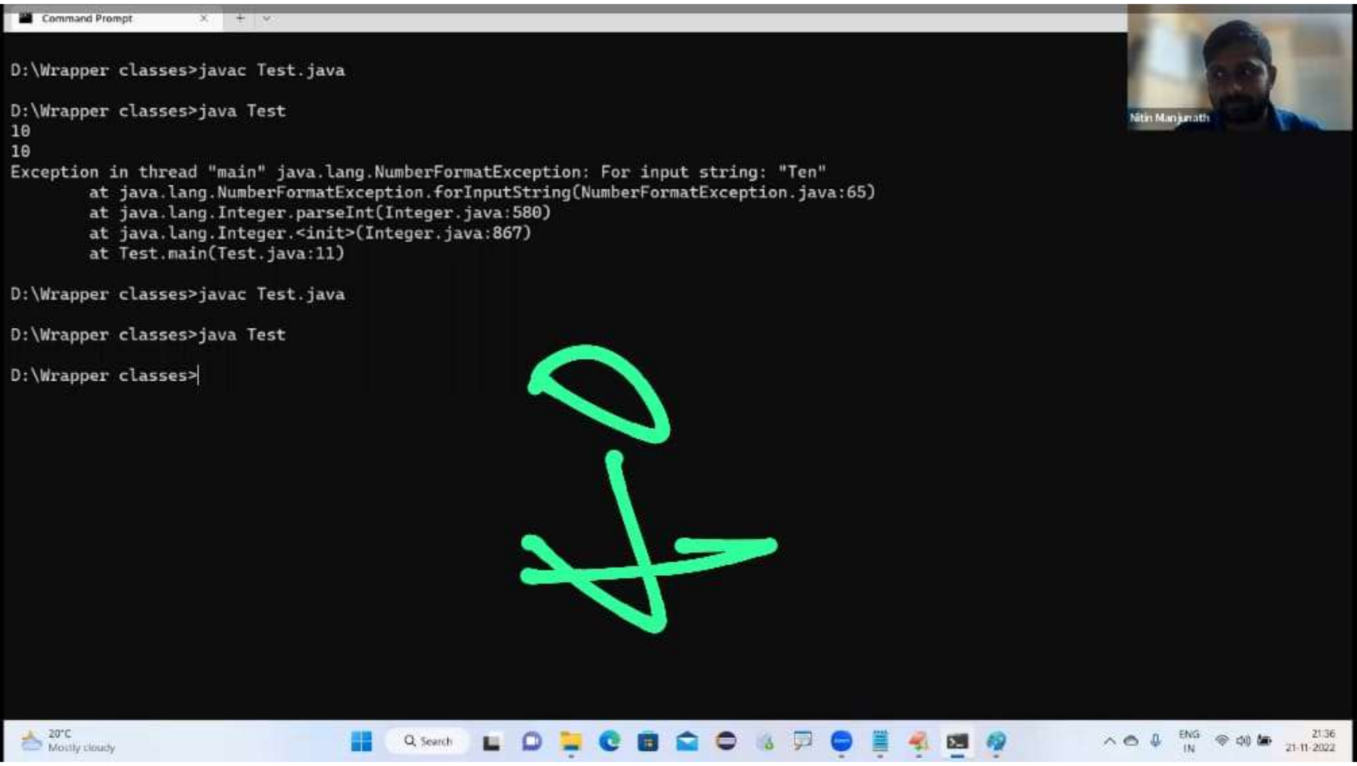
37



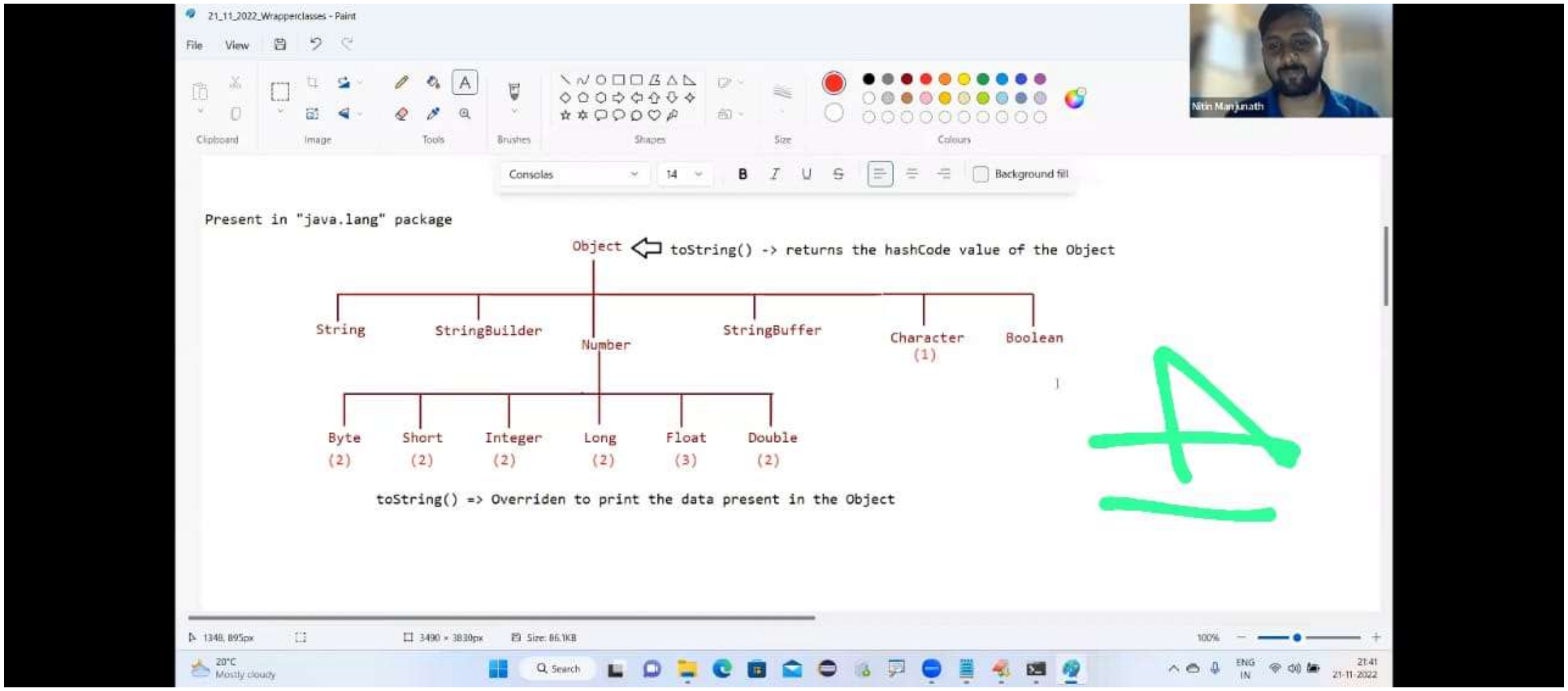
```
  Double d=new Double("10.5");

Note: If String argument is not properly defined then it would result in RunTimeException called
      "NumberformatException".
    eg:: Integer i=new Integer("ten");//RE:NumberFormatException

Wrapper class and its associated constructor
  Byte   => byte and String
  Short  => short and String
 Integer => int and String
 Long    => long and String
 **Float => float ,String and double
  Double => double and String
 **Character=> character
 ***Boolean  => boolean and String
```


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
Java (*.java)
Test.java
For Help, press F1
In 11 col 6 13 00 PC ANSI
Nitin Manjunath
39

```java
class Test
{
    public static void main(String[] args)
    {
        Float f1 = new Float(10.5f);
        Float f2 = new Float("10.5f");
        Float f3 = new Float("10.5");
        Float f4 = new Float(10.5);


    }
}
```



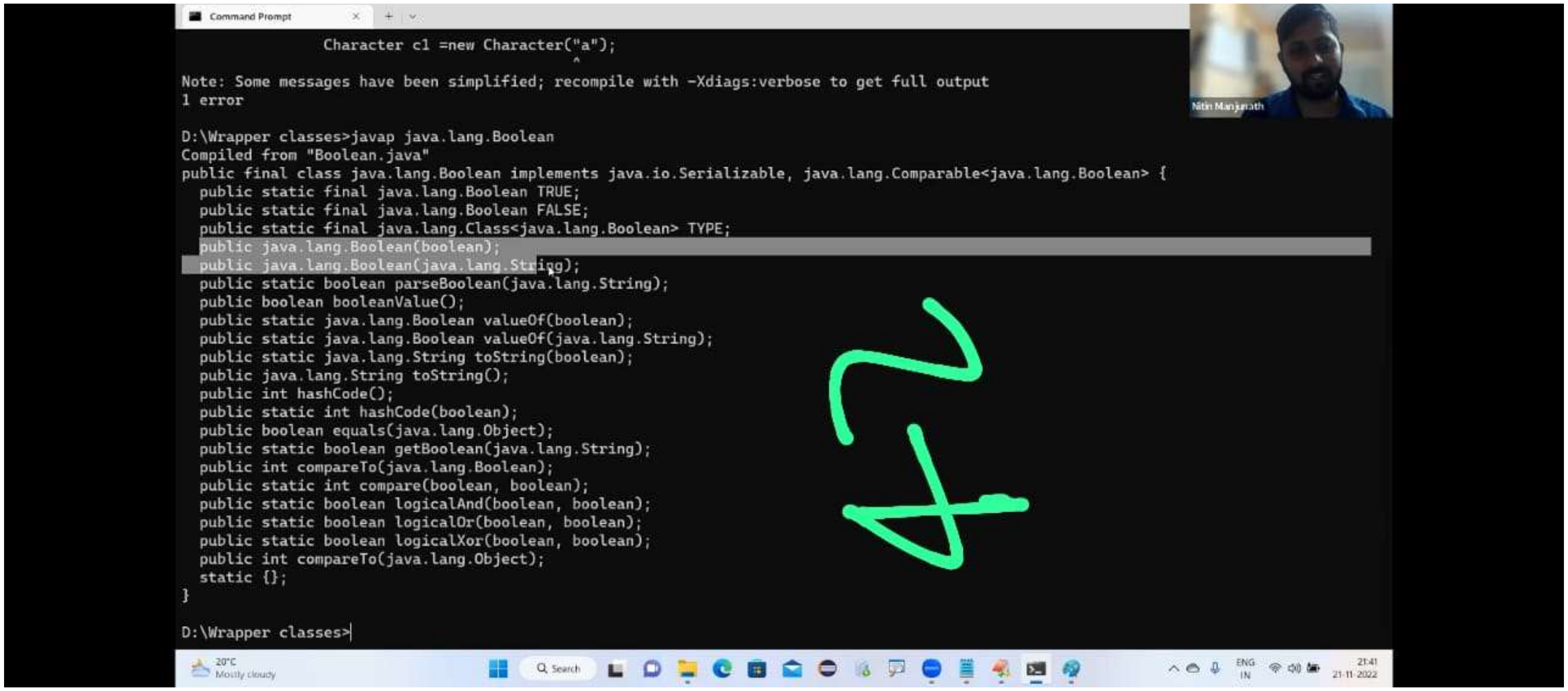
```
D:\Wrapper classes>javac Test.java

D:\Wrapper classes>java Test
10
10
Exception in thread "main" java.lang.NumberFormatException: For input string: "Ten"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.<init>(Integer.java:867)
        at Test.main(Test.java:11)

D:\Wrapper classes>javac Test.java

D:\Wrapper classes>java Test

D:\Wrapper classes>
```



Present in "java.lang" package

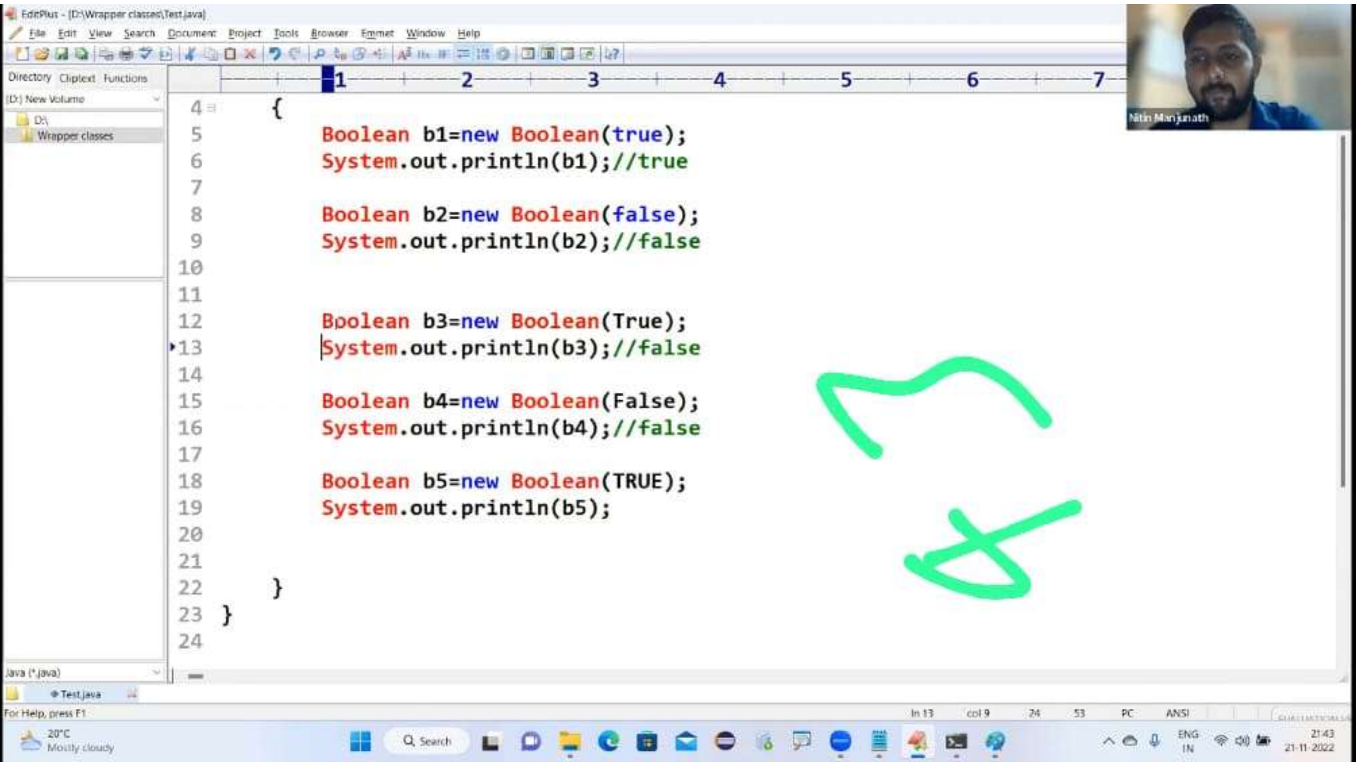
Object ⇦ toString() -> returns the hashCode value of the Object

toString() => Overriden to print the data present in the Object

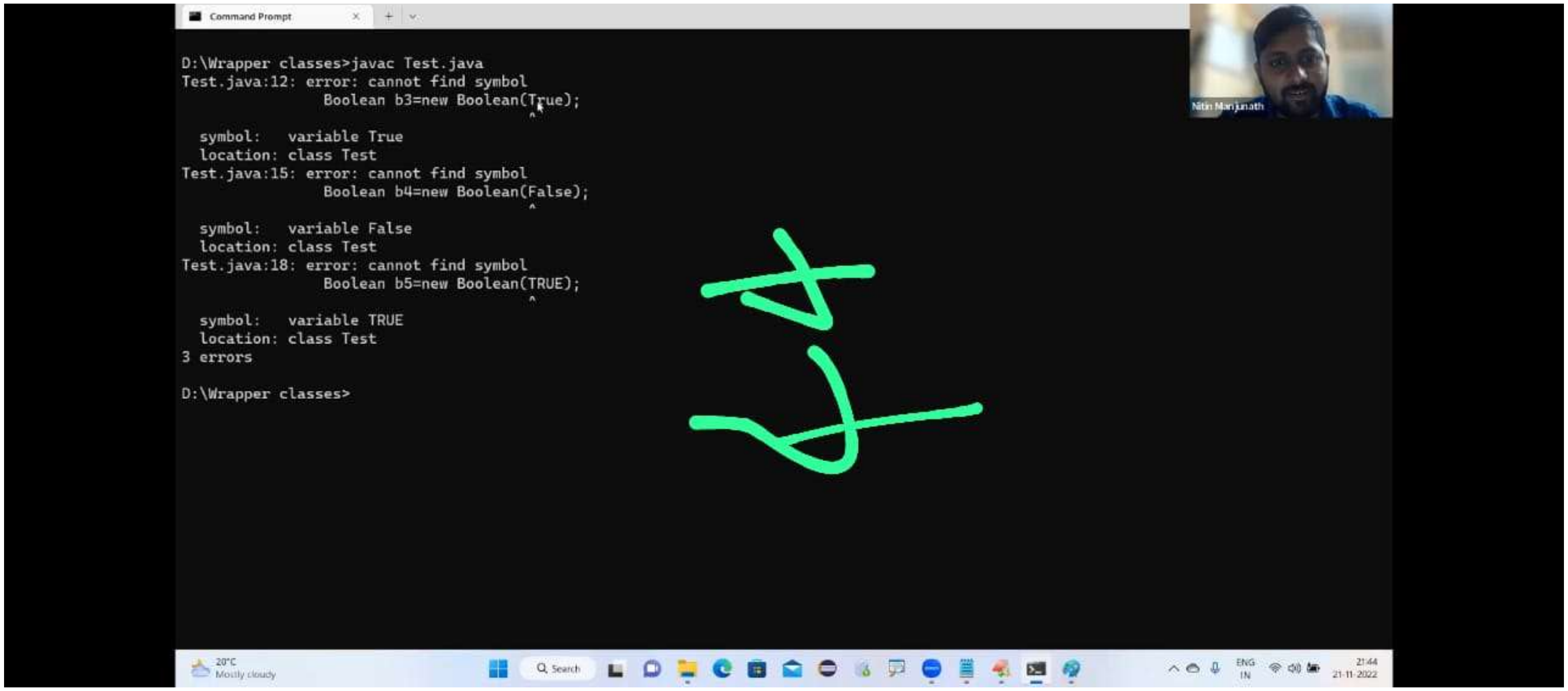


```
                Character c1 =new Character("a");
                                             ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error

D:\Wrapper classes>javap java.lang.Boolean
Compiled from "Boolean.java"
public final class java.lang.Boolean implements java.io.Serializable, java.lang.Comparable<java.lang.Boolean> {
  public static final java.lang.Boolean TRUE;
  public static final java.lang.Boolean FALSE;
  public static final java.lang.Class<java.lang.Boolean> TYPE;
  public java.lang.Boolean(boolean);
  public java.lang.Boolean(java.lang.String);
  public static boolean parseBoolean(java.lang.String);
  public boolean booleanValue();
  public static java.lang.Boolean valueOf(boolean);
  public static java.lang.Boolean valueOf(java.lang.String);
  public static java.lang.String toString(boolean);
  public java.lang.String toString();
  public int hashCode();
  public static int hashCode(boolean);
  public boolean equals(java.lang.Object);
  public static boolean getBoolean(java.lang.String);
  public int compareTo(java.lang.Boolean);
  public static int compare(boolean, boolean);
  public static boolean logicalAnd(boolean, boolean);
  public static boolean logicalOr(boolean, boolean);
  public static boolean logicalXor(boolean, boolean);
  public int compareTo(java.lang.Object);
  static {};
}

D:\Wrapper classes>
```

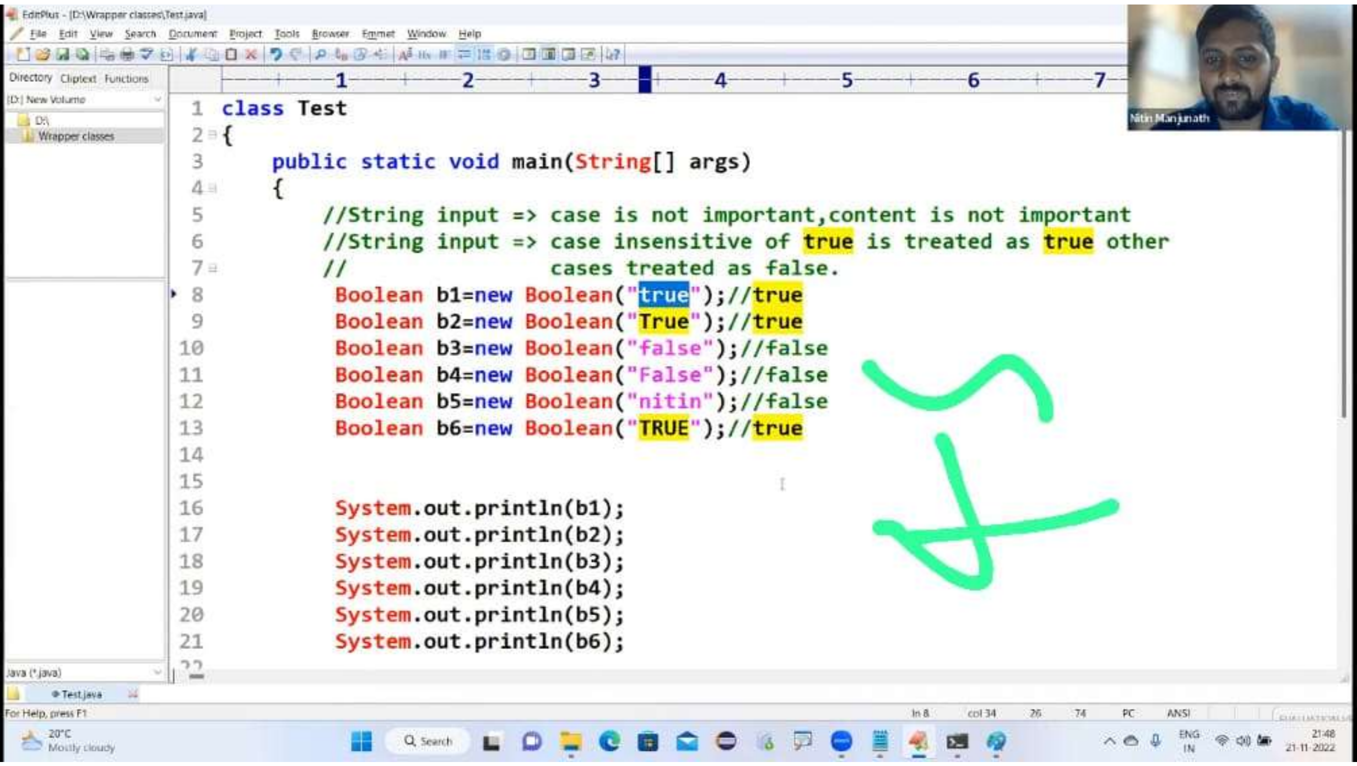
42



```java
    {
        Boolean b1=new Boolean(true);
        System.out.println(b1);//true

        Boolean b2=new Boolean(false);
        System.out.println(b2);//false


        Boolean b3=new Boolean(True);
        System.out.println(b3);//false

        Boolean b4=new Boolean(False);
        System.out.println(b4);//false

        Boolean b5=new Boolean(TRUE);
        System.out.println(b5);


    }
}
```



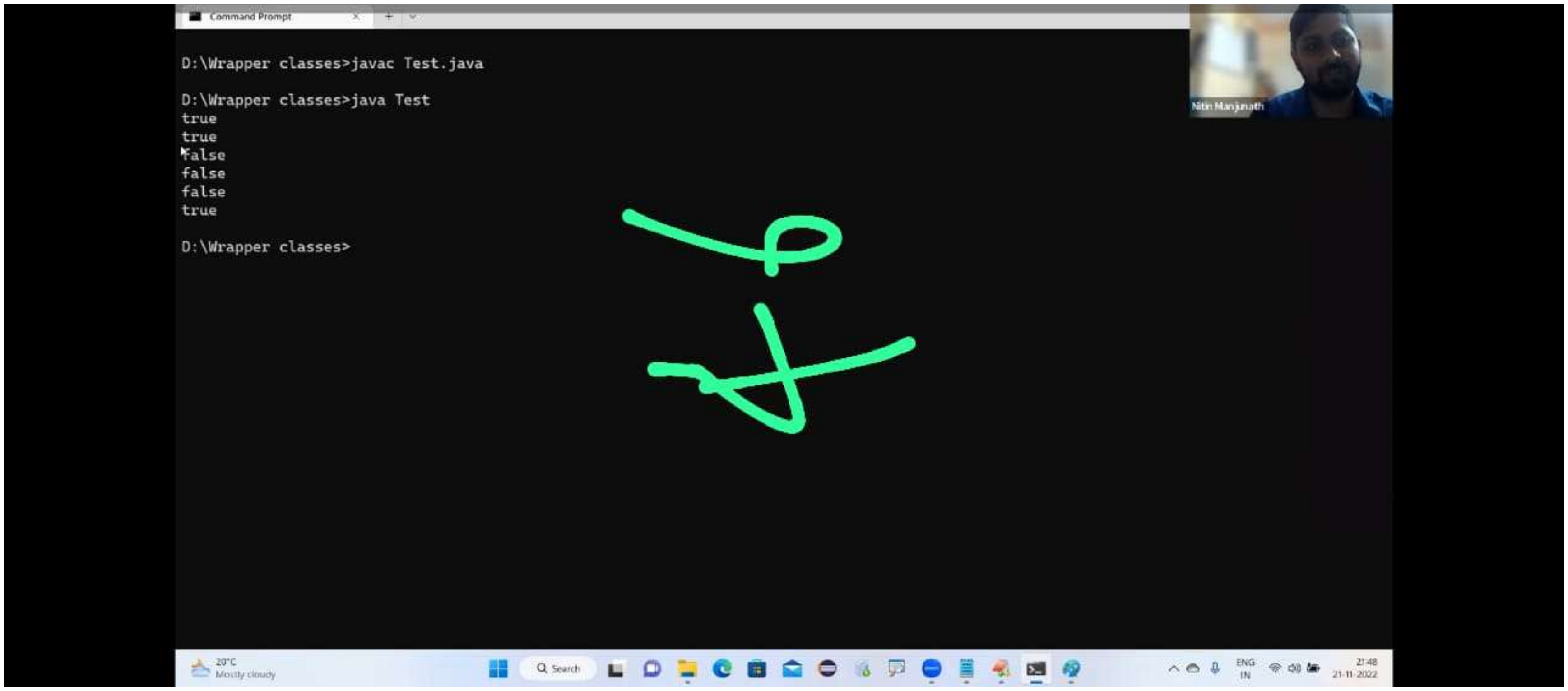
```
D:\Wrapper classes>javac Test.java
Test.java:12: error: cannot find symbol
                Boolean b3=new Boolean(True);
                                       ^
  symbol:   variable True
  location: class Test
Test.java:15: error: cannot find symbol
                Boolean b4=new Boolean(False);
                                       ^
  symbol:   variable False
  location: class Test
Test.java:18: error: cannot find symbol
                Boolean b5=new Boolean(TRUE);
                                       ^
  symbol:   variable TRUE
  location: class Test
3 errors

D:\Wrapper classes>
```



EditPlus - [D:\Wrapper classes\Test.java]

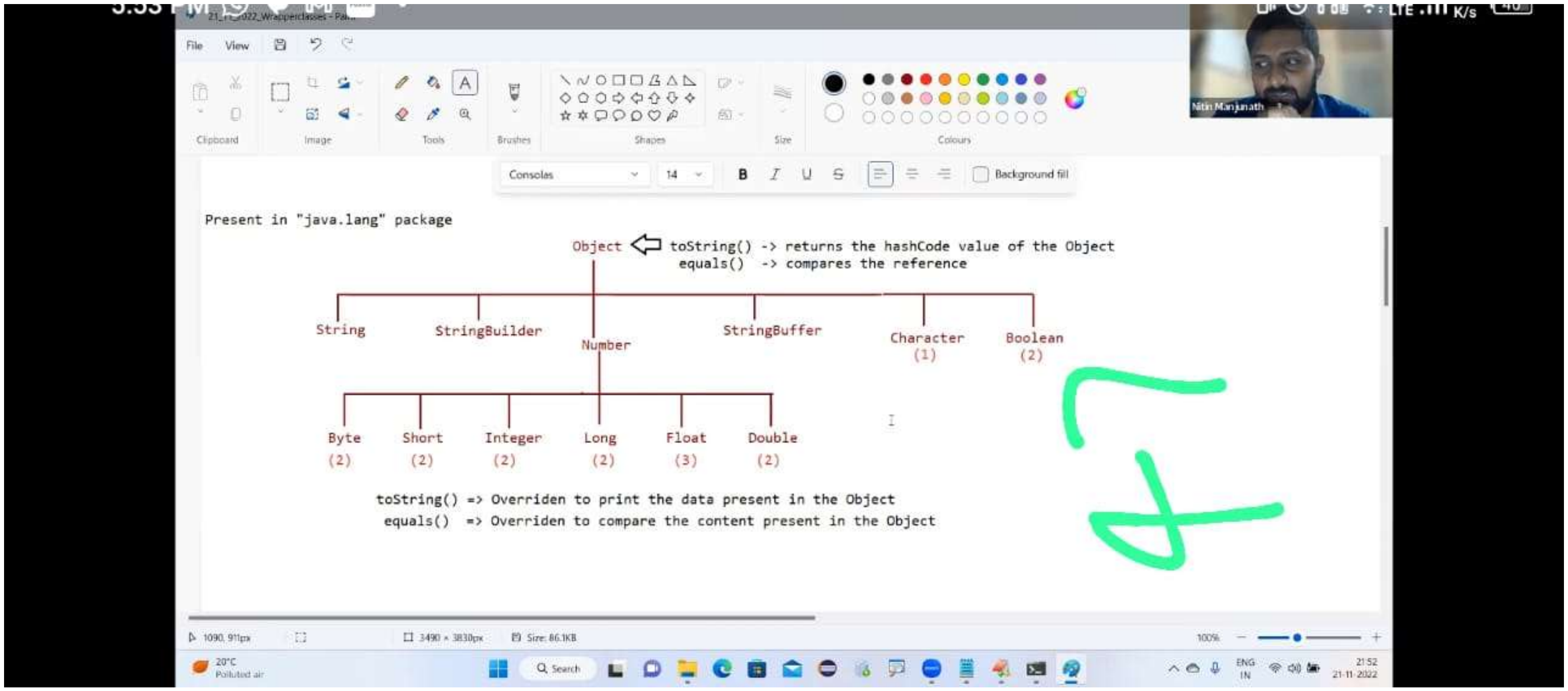
```java
class Test
{
    public static void main(String[] args)
    {
        //String input => case is not important,content is not important
        //String input => case insensitive of true is treated as true other
        //                cases treated as false.
         Boolean b1=new Boolean("true");//true
         Boolean b2=new Boolean("True");//true
         Boolean b3=new Boolean("false");//false
         Boolean b4=new Boolean("False");//false
         Boolean b5=new Boolean("nitin");//false
         Boolean b6=new Boolean("TRUE");//true


         System.out.println(b1);
         System.out.println(b2);
         System.out.println(b3);
         System.out.println(b4);
         System.out.println(b5);
         System.out.println(b6);
```

45



```
D:\Wrapper classes>javac Test.java

D:\Wrapper classes>java Test
true
true
false
false
false
true

D:\Wrapper classes>
```

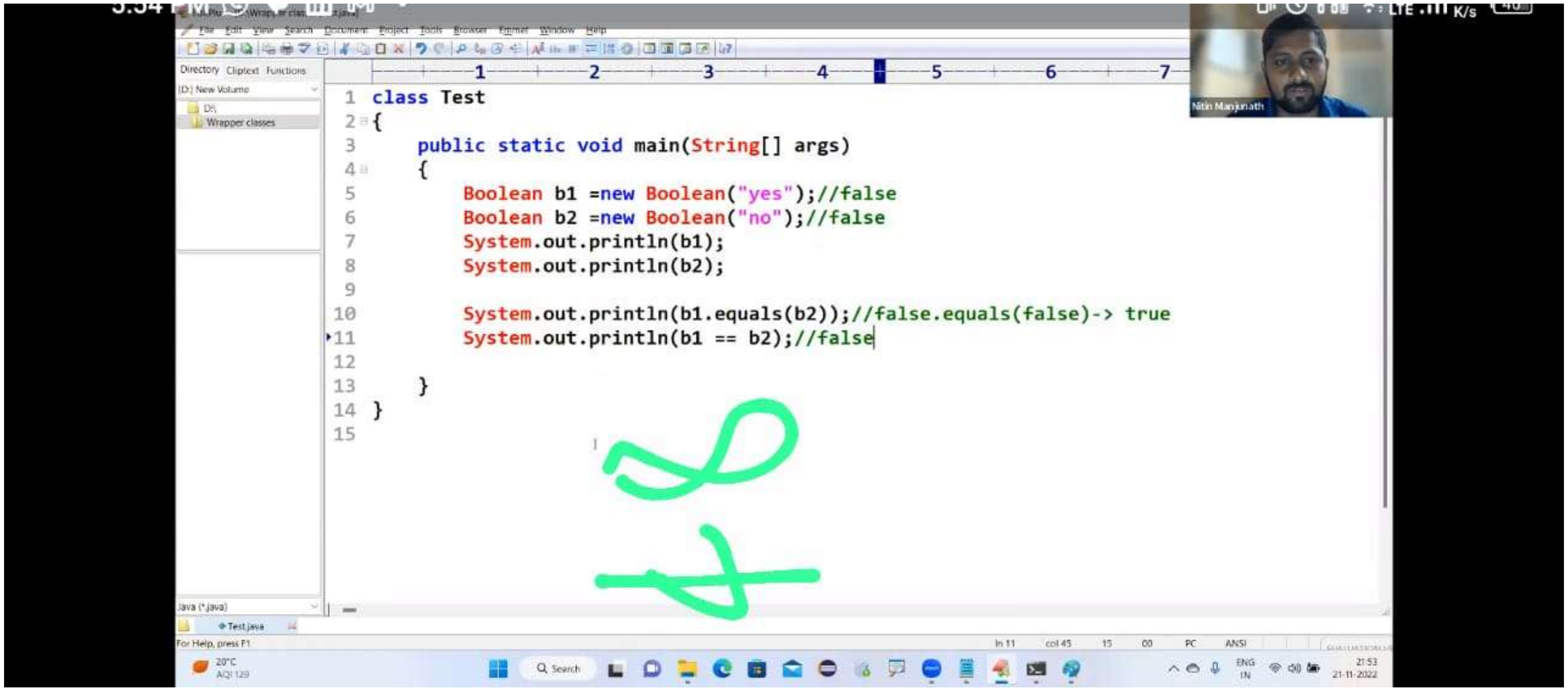
46



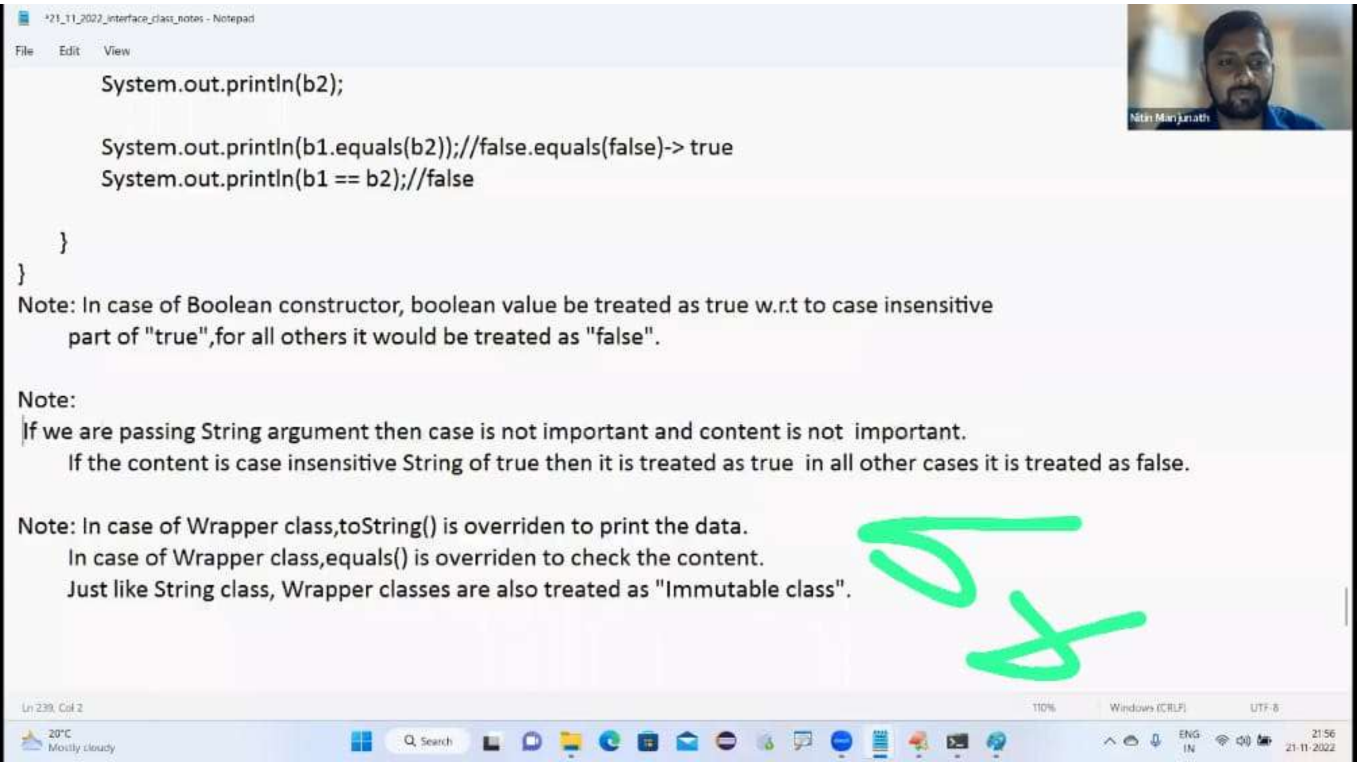
Present in "java.lang" package

toString() => Overriden to print the data present in the Object
equals() => Overriden to compare the content present in the Object

47



```java
class Test
{
    public static void main(String[] args)
    {
        Boolean b1 =new Boolean("yes");//false
        Boolean b2 =new Boolean("no");//false
        System.out.println(b1);
        System.out.println(b2);

        System.out.println(b1.equals(b2));//false.equals(false)-> true
        System.out.println(b1 == b2);//false

    }
}
```



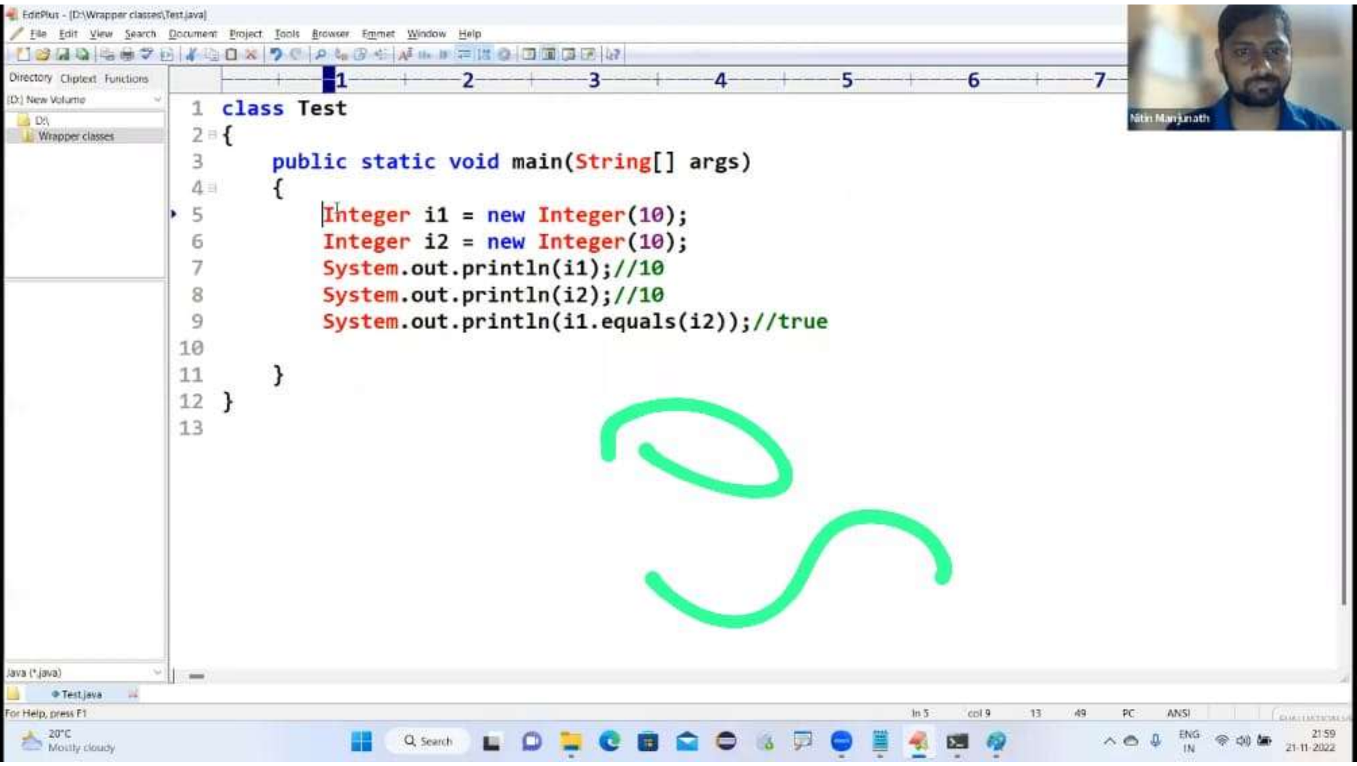
```
        System.out.println(b2);

        System.out.println(b1.equals(b2));//false.equals(false)-> true
        System.out.println(b1 == b2);//false

    }
}
```

Note: In case of Boolean constructor, boolean value be treated as true w.r.t to case insensitive
part of "true",for all others it would be treated as "false".

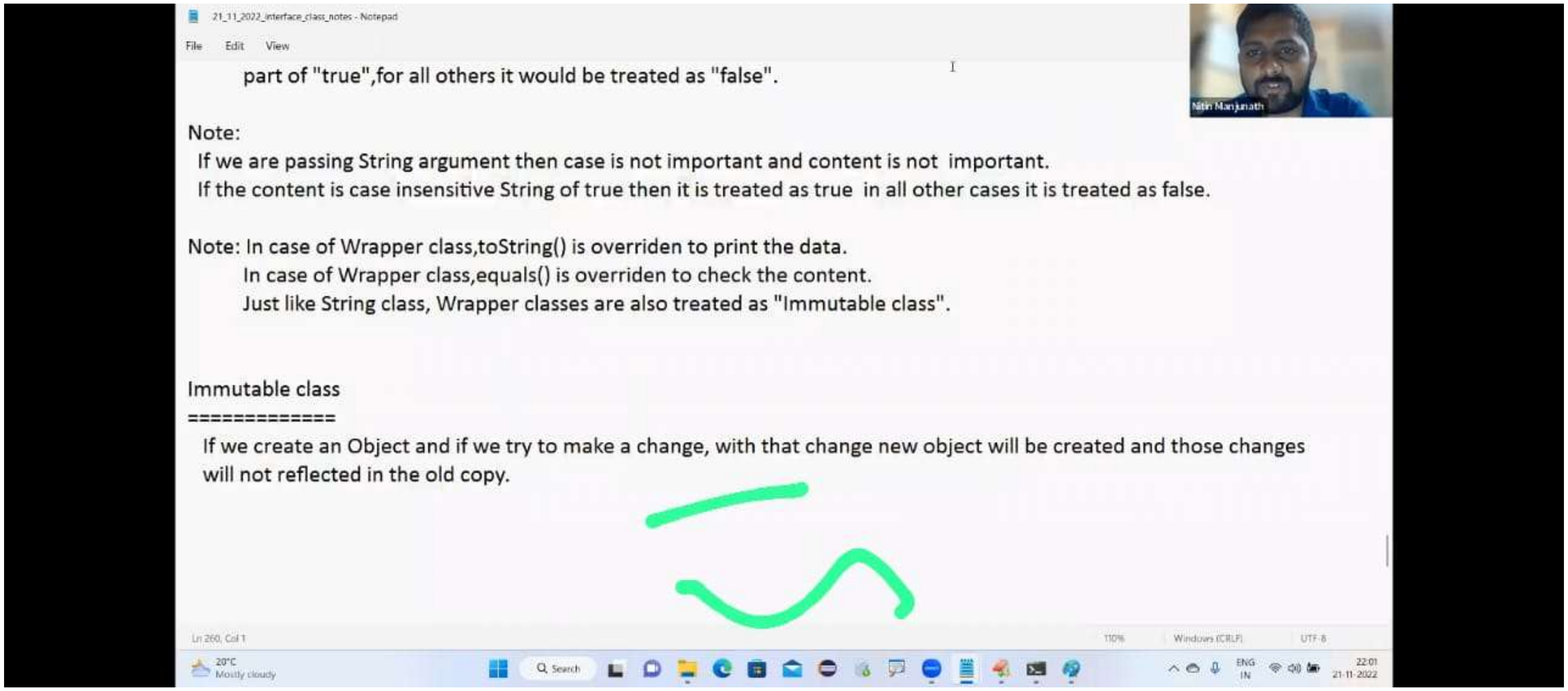
Note:
If we are passing String argument then case is not important and content is not important.
If the content is case insensitive String of true then it is treated as true in all other cases it is treated as false.

Note: In case of Wrapper class,toString() is overriden to print the data.
In case of Wrapper class,equals() is overriden to check the content.
Just like String class, Wrapper classes are also treated as "Immutable class".

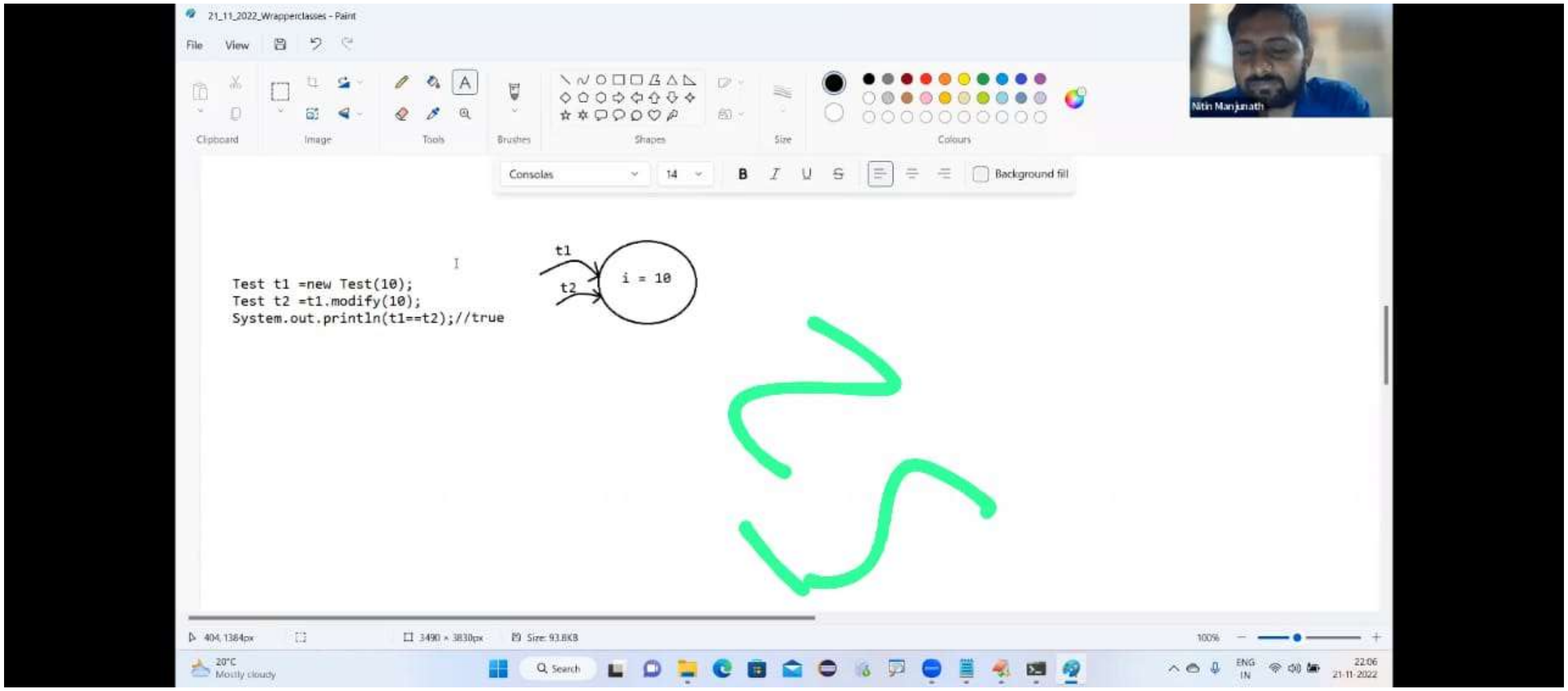
49



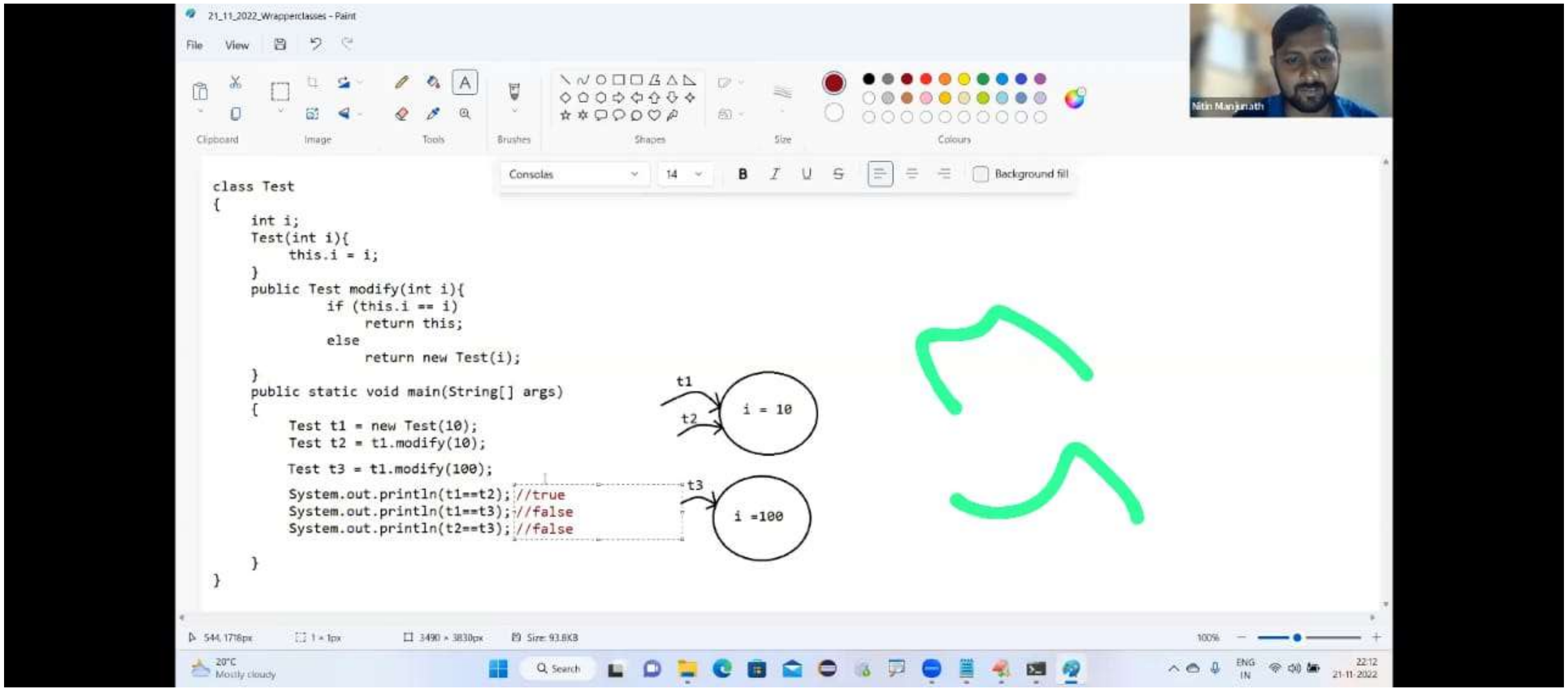
```java
class Test
{
    public static void main(String[] args)
    {
        Integer i1 = new Integer(10);
        Integer i2 = new Integer(10);
        System.out.println(i1);//10
        System.out.println(i2);//10
        System.out.println(i1.equals(i2));//true

    }
}
```



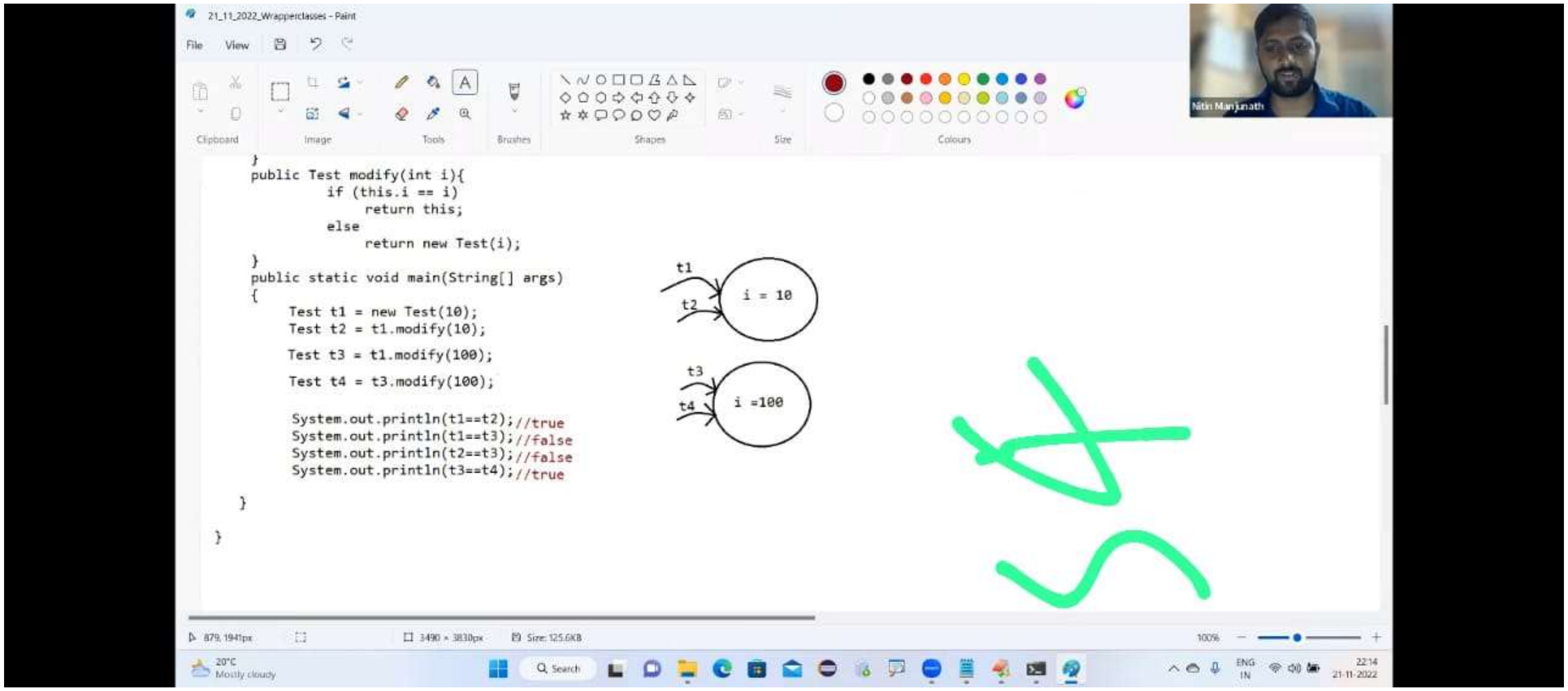
```
        part of "true",for all others it would be treated as "false".

Note:
 If we are passing String argument then case is not important and content is not  important.
 If the content is case insensitive String of true then it is treated as true  in all other cases it is treated as false.

Note: In case of Wrapper class,toString() is overriden to print the data.
        In case of Wrapper class,equals() is overriden to check the content.
        Just like String class, Wrapper classes are also treated as "Immutable class".


Immutable class
=============
  If we create an Object and if we try to make a change, with that change new object will be created and those changes
  will not reflected in the old copy.
```



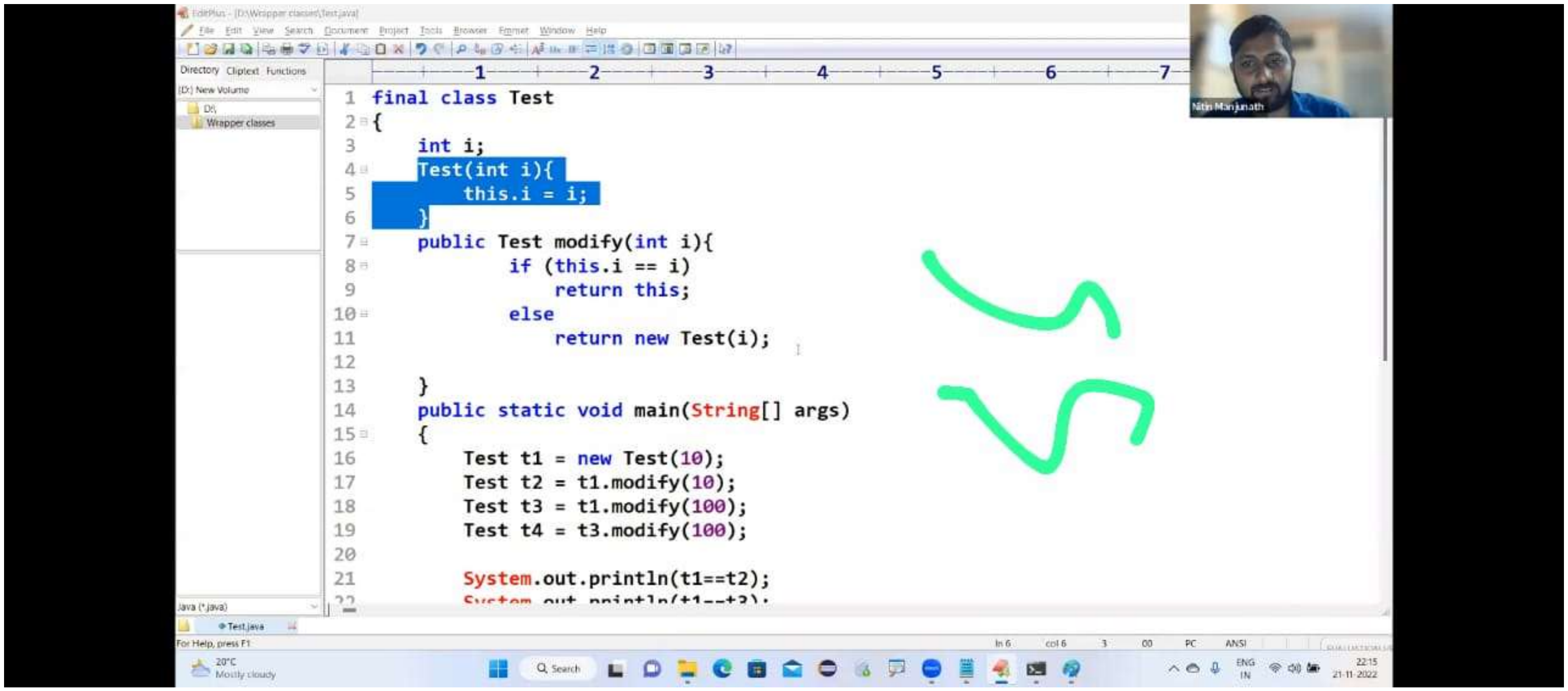
```
Test t1 =new Test(10);
Test t2 =t1.modify(10);
System.out.println(t1==t2);//true
```



```
class Test
{
    int i;
    Test(int i){
        this.i = i;
    }
    public Test modify(int i){
            if (this.i == i)
                return this;
            else
                return new Test(i);
    }
    public static void main(String[] args)
    {
        Test t1 = new Test(10);
        Test t2 = t1.modify(10);

        Test t3 = t1.modify(100);

        System.out.println(t1==t2); //true
        System.out.println(t1==t3); //false
        System.out.println(t2==t3); //false

    }
}
```



```java
    }
    public Test modify(int i){
            if (this.i == i)
                return this;
            else
                return new Test(i);
    }
    public static void main(String[] args)
    {
        Test t1 = new Test(10);
        Test t2 = t1.modify(10);

        Test t3 = t1.modify(100);

        Test t4 = t3.modify(100);

        System.out.println(t1==t2);//true
        System.out.println(t1==t3);//false
        System.out.println(t2==t3);//false
        System.out.println(t3==t4);//true

    }

}
```



```java
final class Test
{
    int i;
    Test(int i){
        this.i = i;
    }
    public Test modify(int i){
            if (this.i == i)
                return this;
            else
                return new Test(i);

    }
    public static void main(String[] args)
    {
        Test t1 = new Test(10);
        Test t2 = t1.modify(10);
        Test t3 = t1.modify(100);
        Test t4 = t3.modify(100);

        System.out.println(t1==t2);
        System.out.println(t1==t3);
```