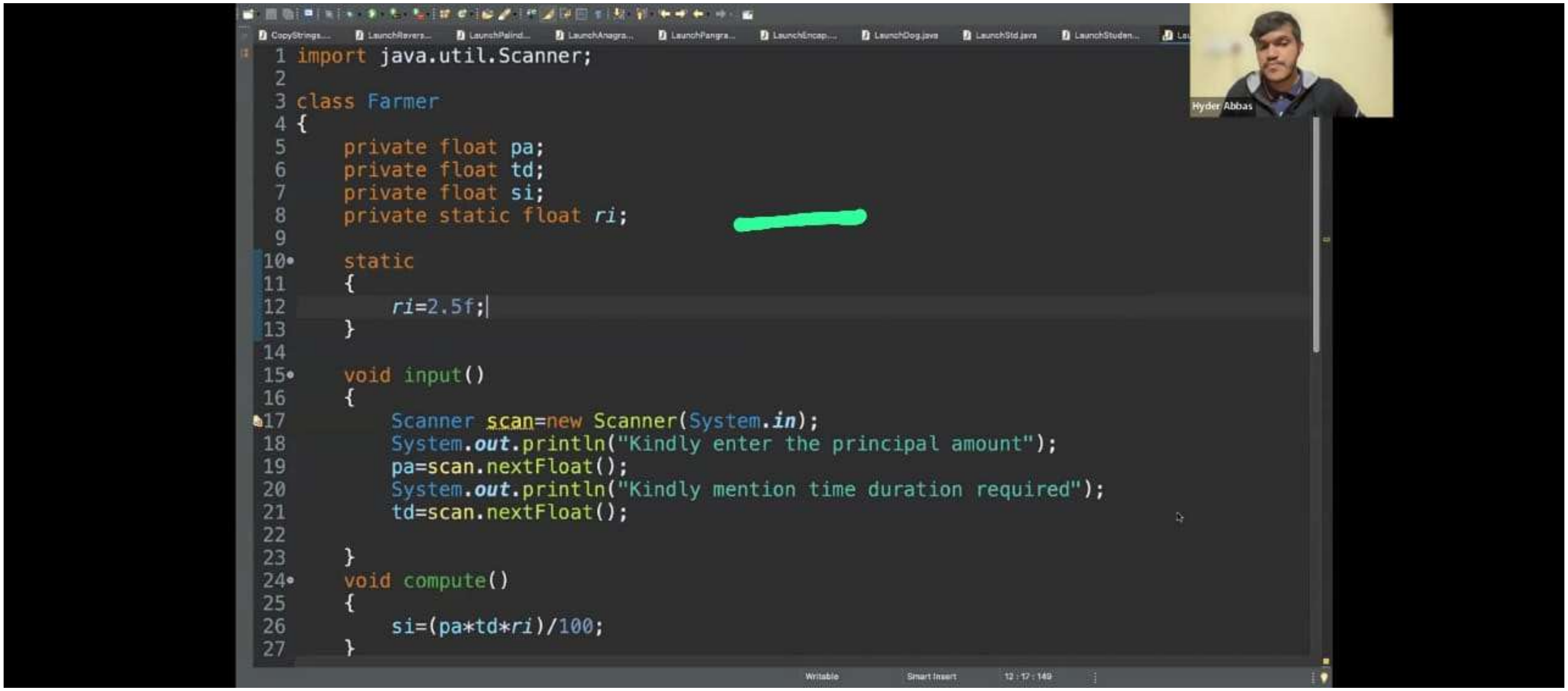
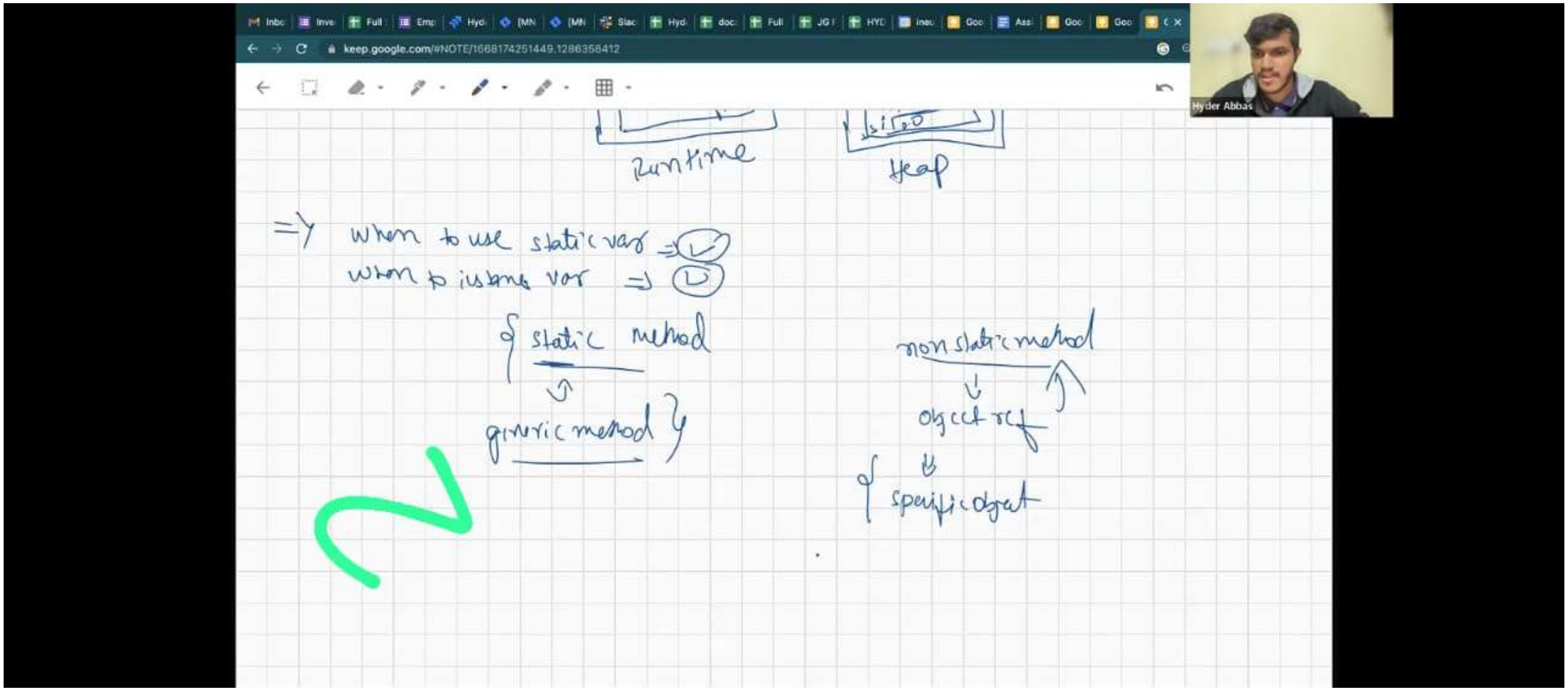
Static in Java



```java
import java.util.Scanner;

class Farmer
{
    private float pa;
    private float td;
    private float si;
    private static float ri;

    static
    {
        ri=2.5f;
    }

    void input()
    {
        Scanner scan=new Scanner(System.in);
        System.out.println("Kindly enter the principal amount");
        pa=scan.nextFloat();
        System.out.println("Kindly mention time duration required");
        td=scan.nextFloat();

    }
    void compute()
    {
        si=(pa*td*ri)/100;
    }
```

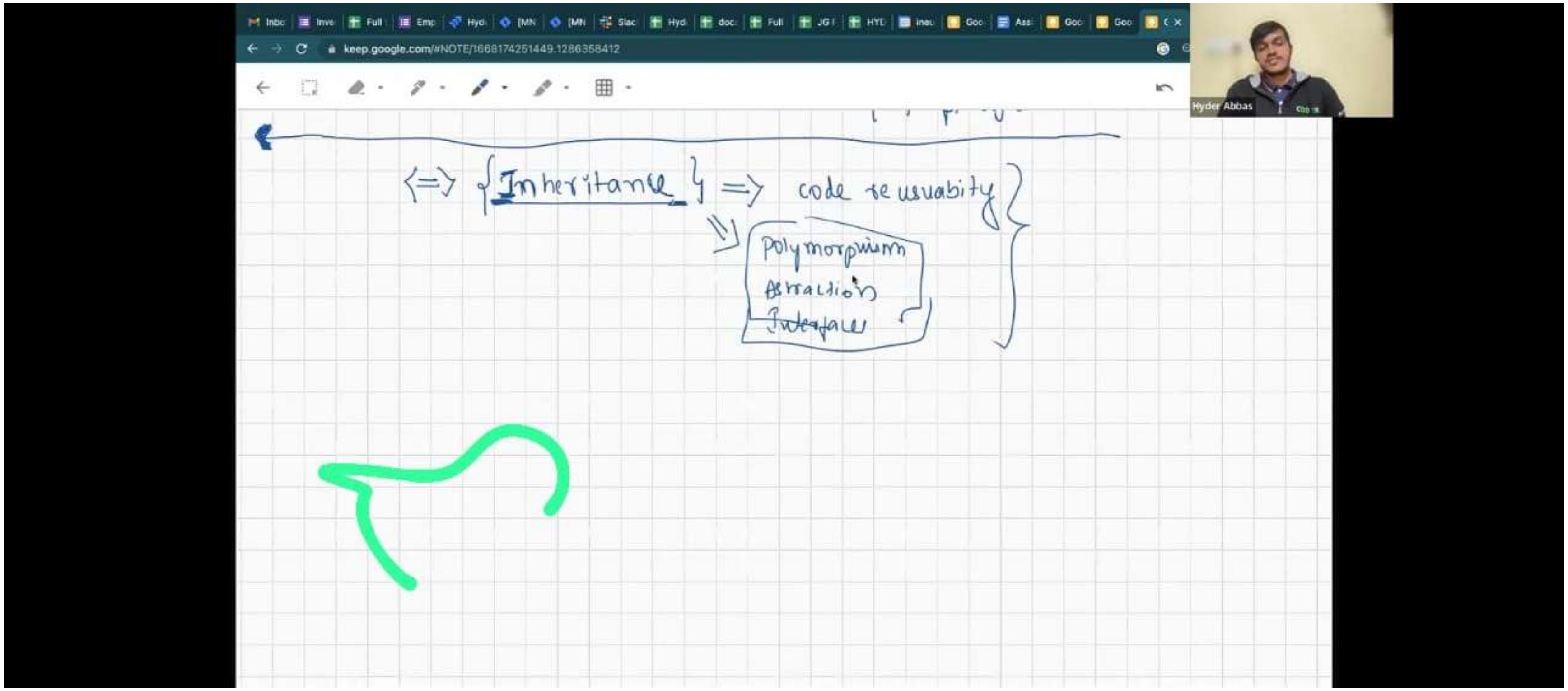


=> When to use static var => L
when to instance var => D

{ static method
↺
generic method }

non static method
↓
object ref
⇓
{ specific object

2



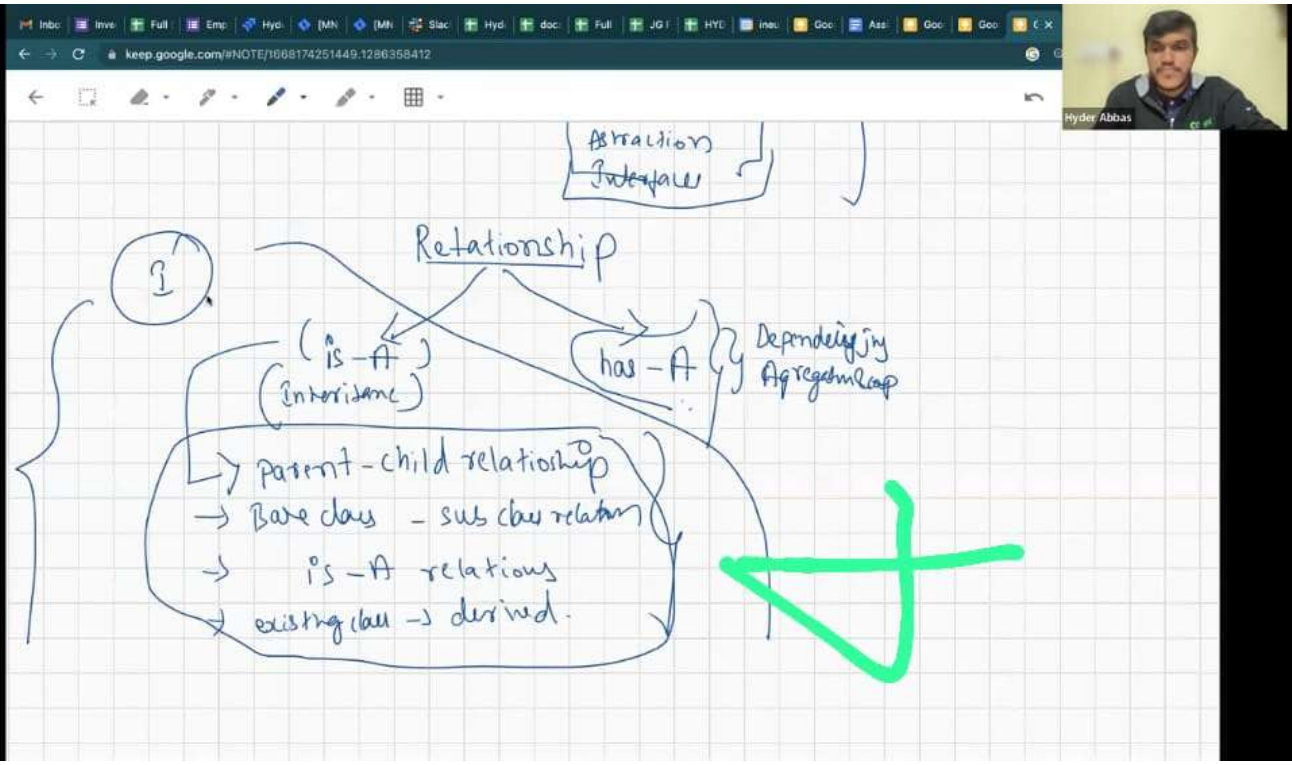
⟨=⟩ {Inheritance} ⟹ code reusabity

⟹

Polymorphism

Abstraction

Interface



Abstraction
Interface

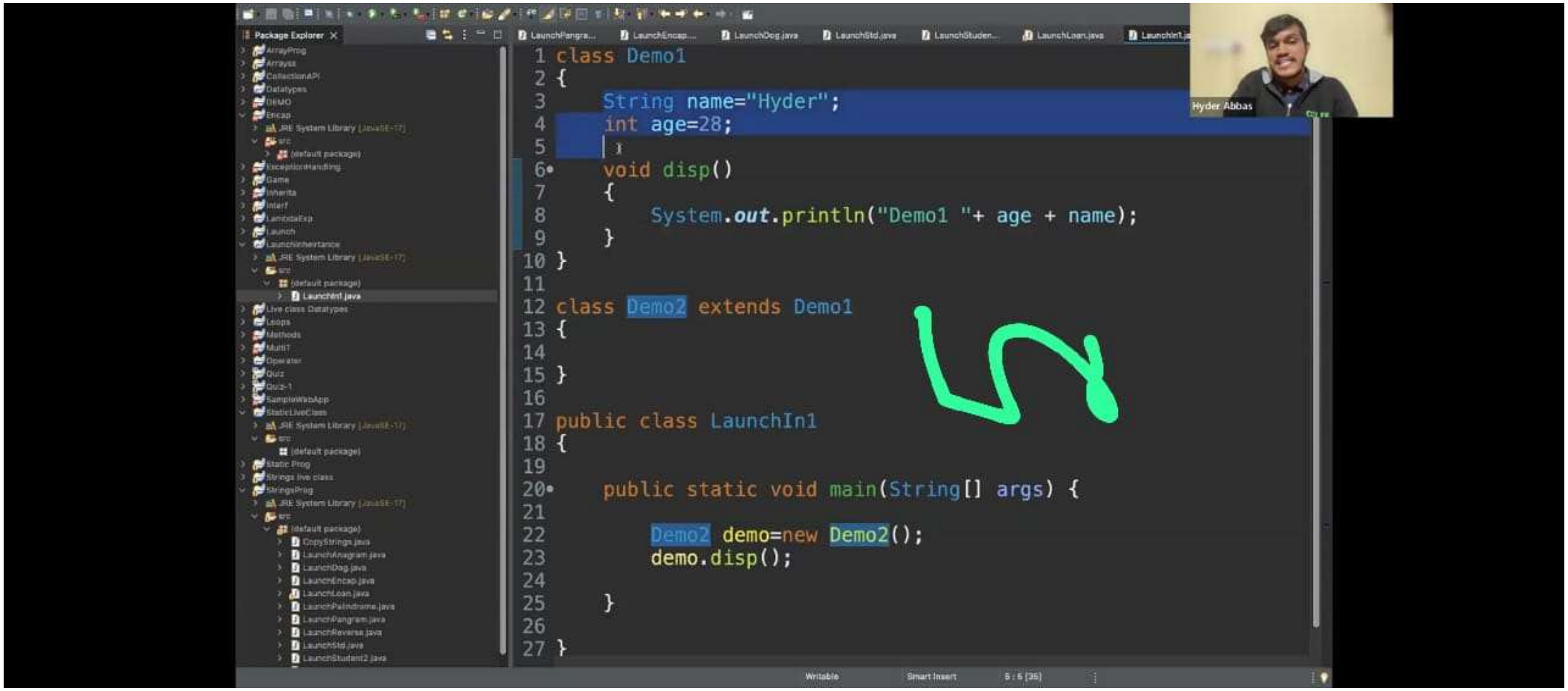
Relationship

1

(is-A)
(Inheritance)

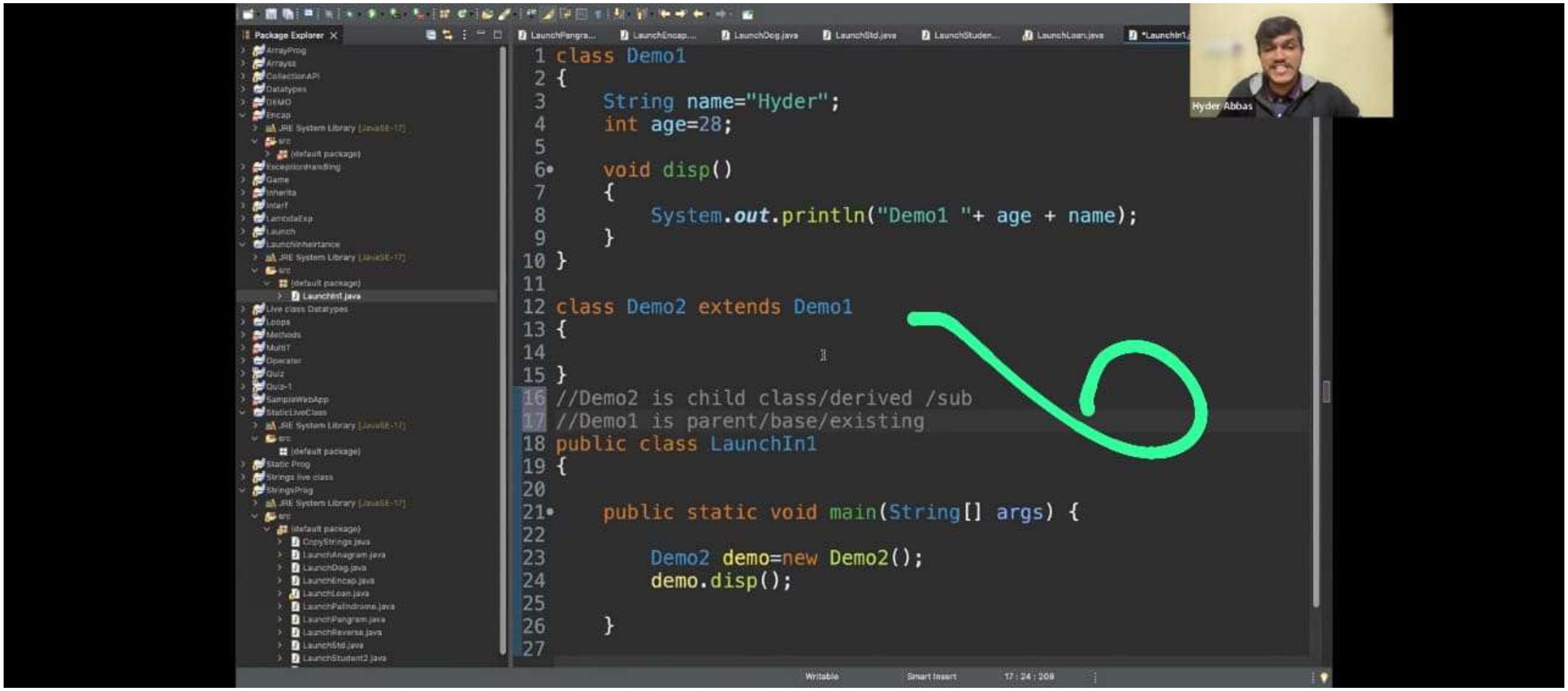
has-A
Dependency
Aggregation

→ Parent-child relationship
→ Base class - sub class relation
→ is-A relations
→ existing class → derived.

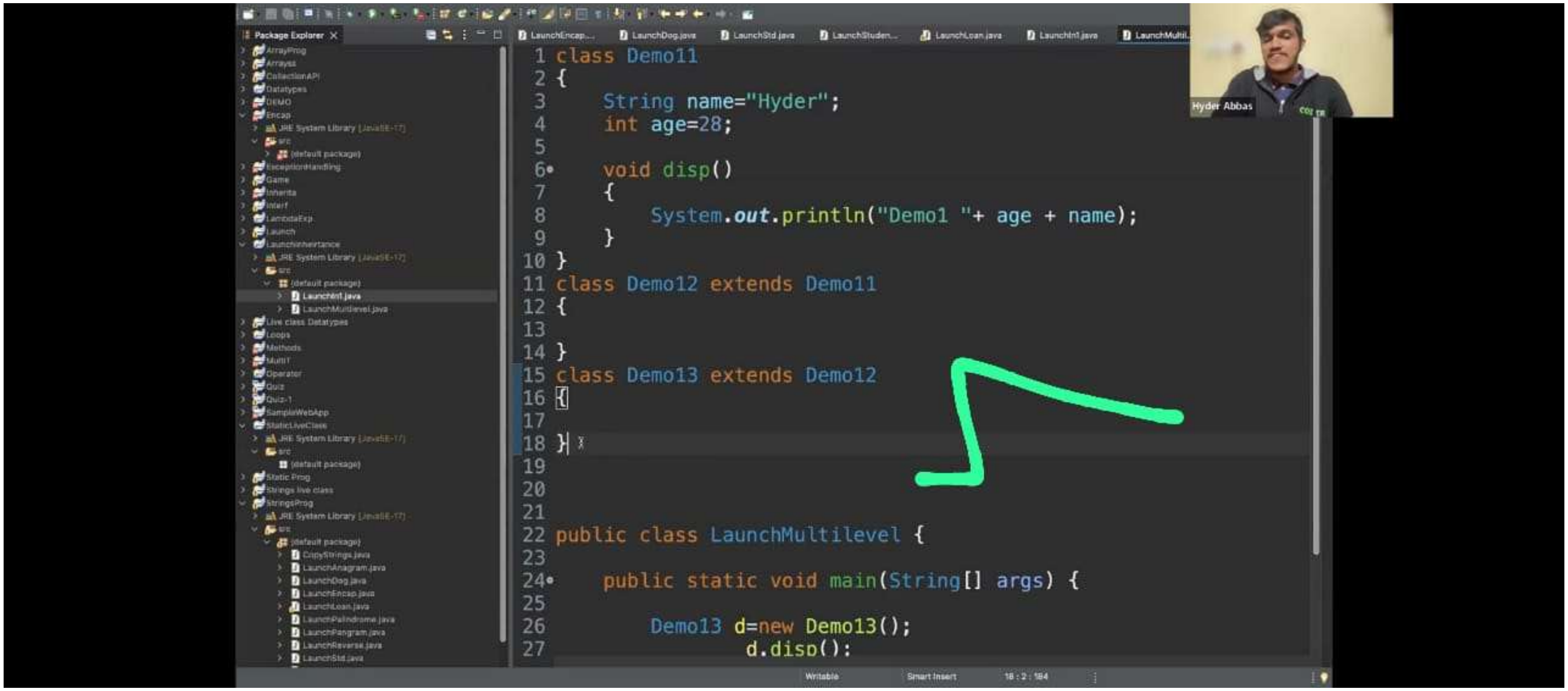


```java
class Demo1
{
    String name="Hyder";
    int age=28;
    
    void disp()
    {
        System.out.println("Demo1 "+ age + name);
    }
}

class Demo2 extends Demo1
{

}

public class LaunchIn1
{

    public static void main(String[] args) {

        Demo2 demo=new Demo2();
        demo.disp();

    }

}
```

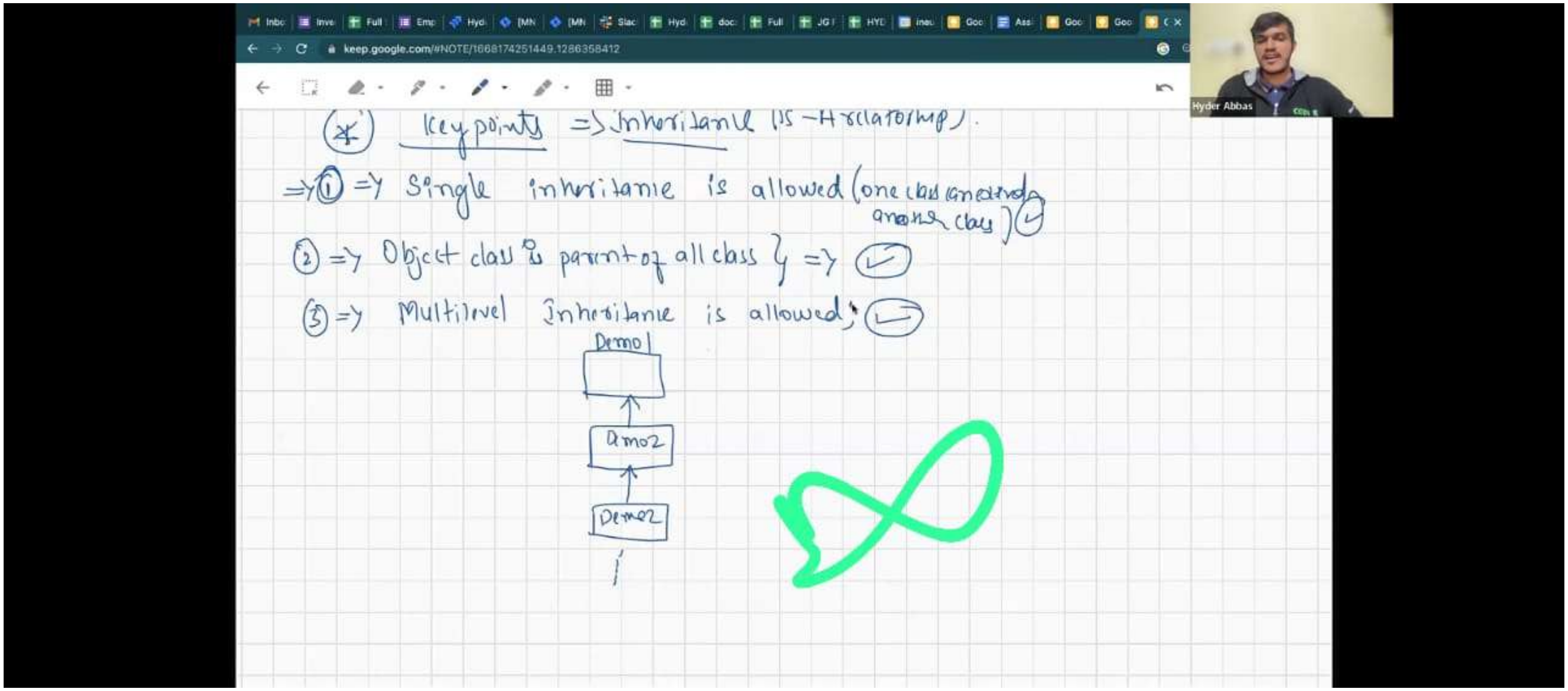
5



```java
class Demo1
{
    String name="Hyder";
    int age=28;

    void disp()
    {
        System.out.println("Demo1 "+ age + name);
    }
}

class Demo2 extends Demo1
{

}
//Demo2 is child class/derived /sub
//Demo1 is parent/base/existing
public class LaunchIn1
{

    public static void main(String[] args) {

        Demo2 demo=new Demo2();
        demo.disp();

    }

```



```java
class Demo11
{
    String name="Hyder";
    int age=28;

    void disp()
    {
        System.out.println("Demo1 "+ age + name);
    }
}
class Demo12 extends Demo11
{

}
class Demo13 extends Demo12
{

}



public class LaunchMultilevel {

    public static void main(String[] args) {

        Demo13 d=new Demo13();
                d.disp();
```

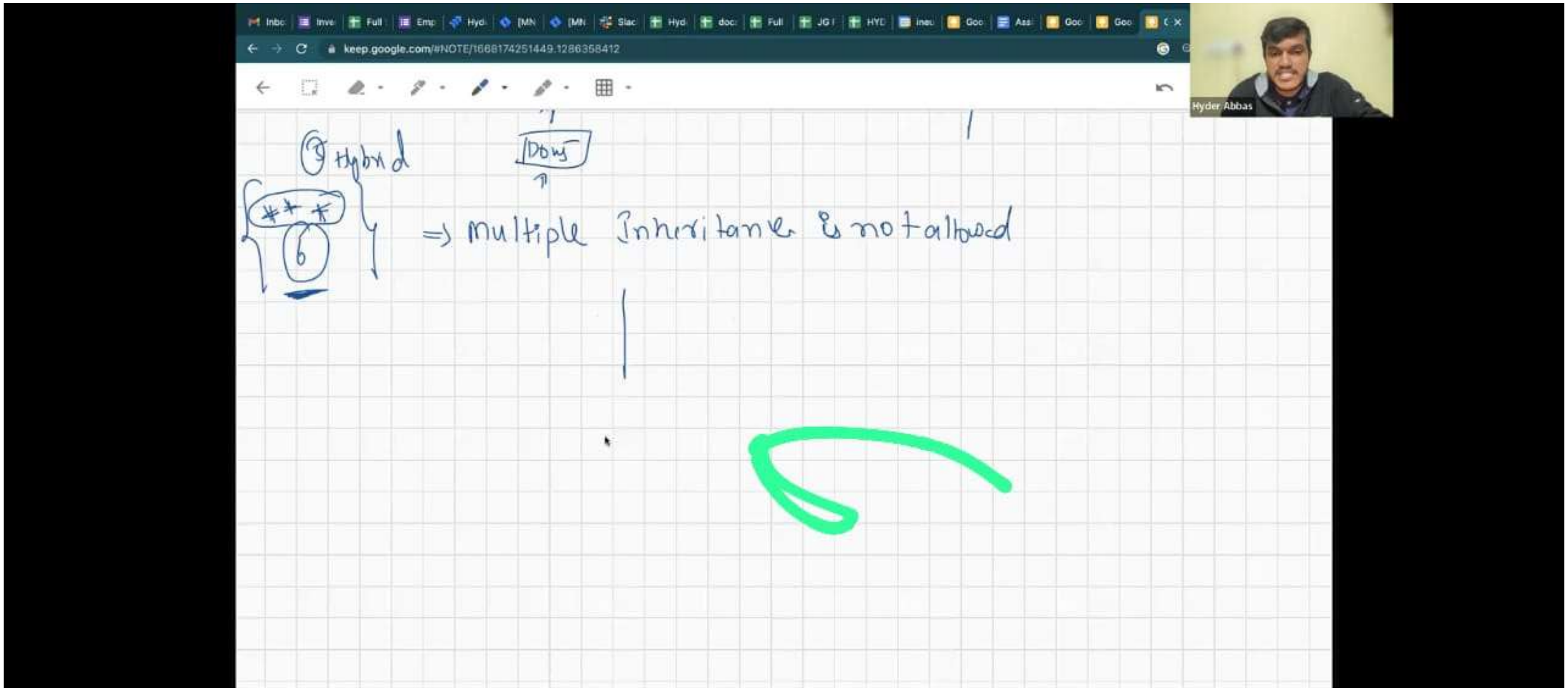


(*) Keypoints => Inheritance (IS-A relationship).

=> ① => Single inheritance is allowed (one class can extends another class) ✓

② => Object class is parent of all class } => ✓

③ => Multilevel Inheritance is allowed; ✓



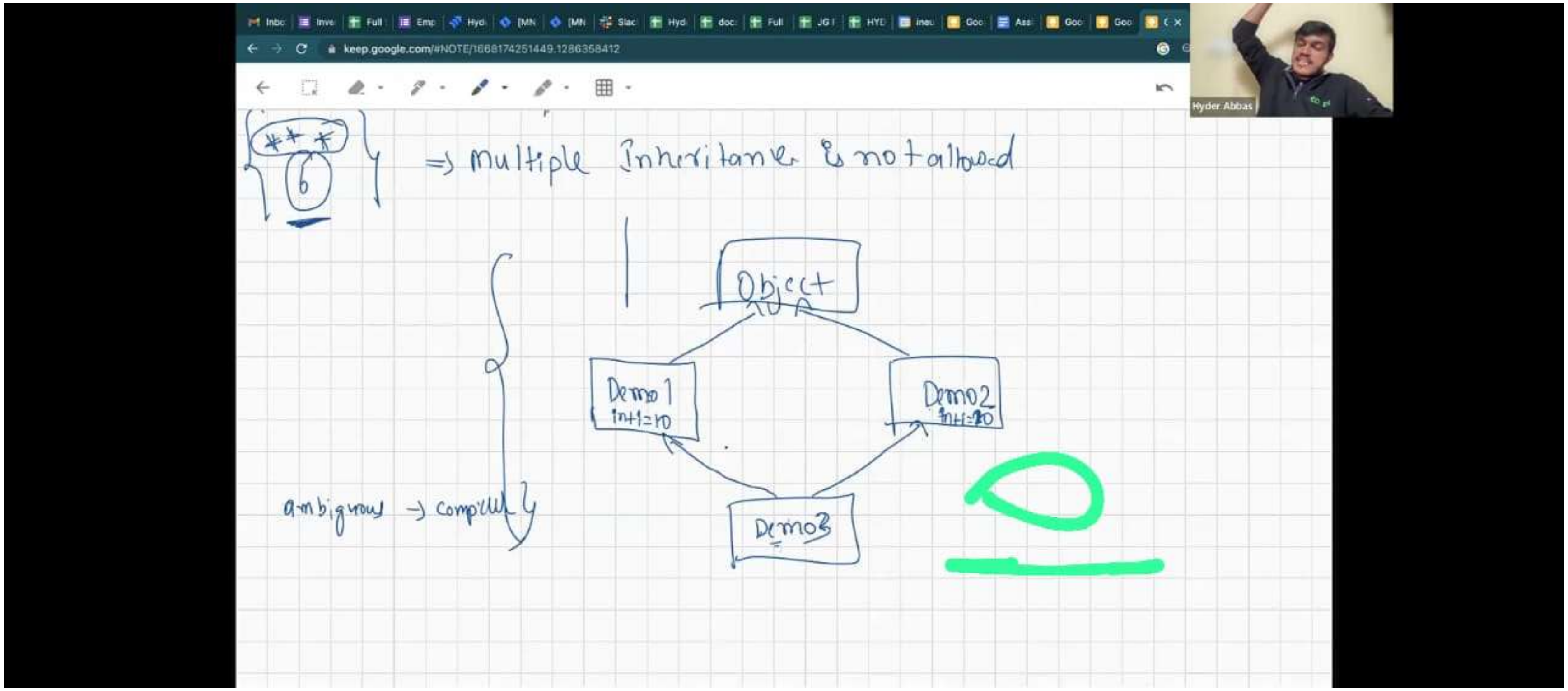
⑤ Hybrid

Dows

***

6

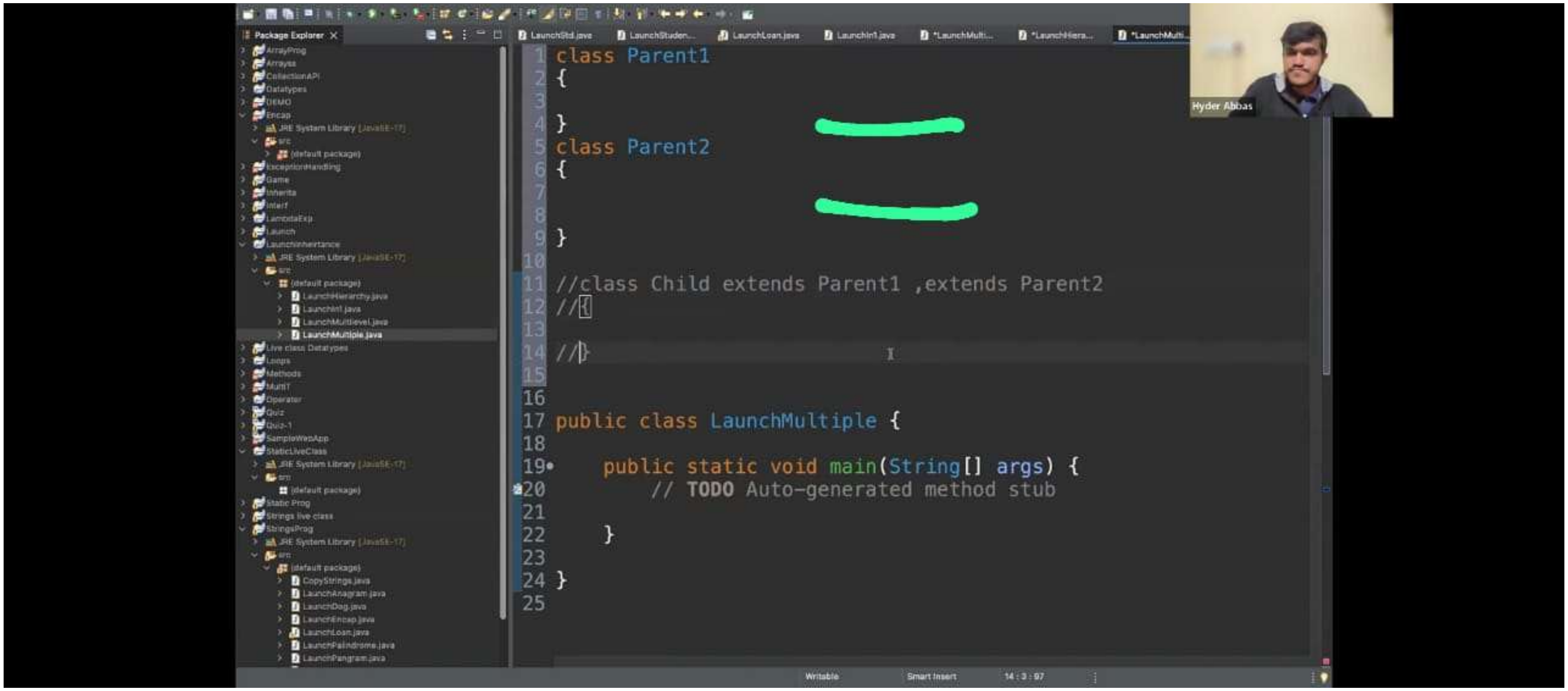
=> multiple Inheritance is not allowed



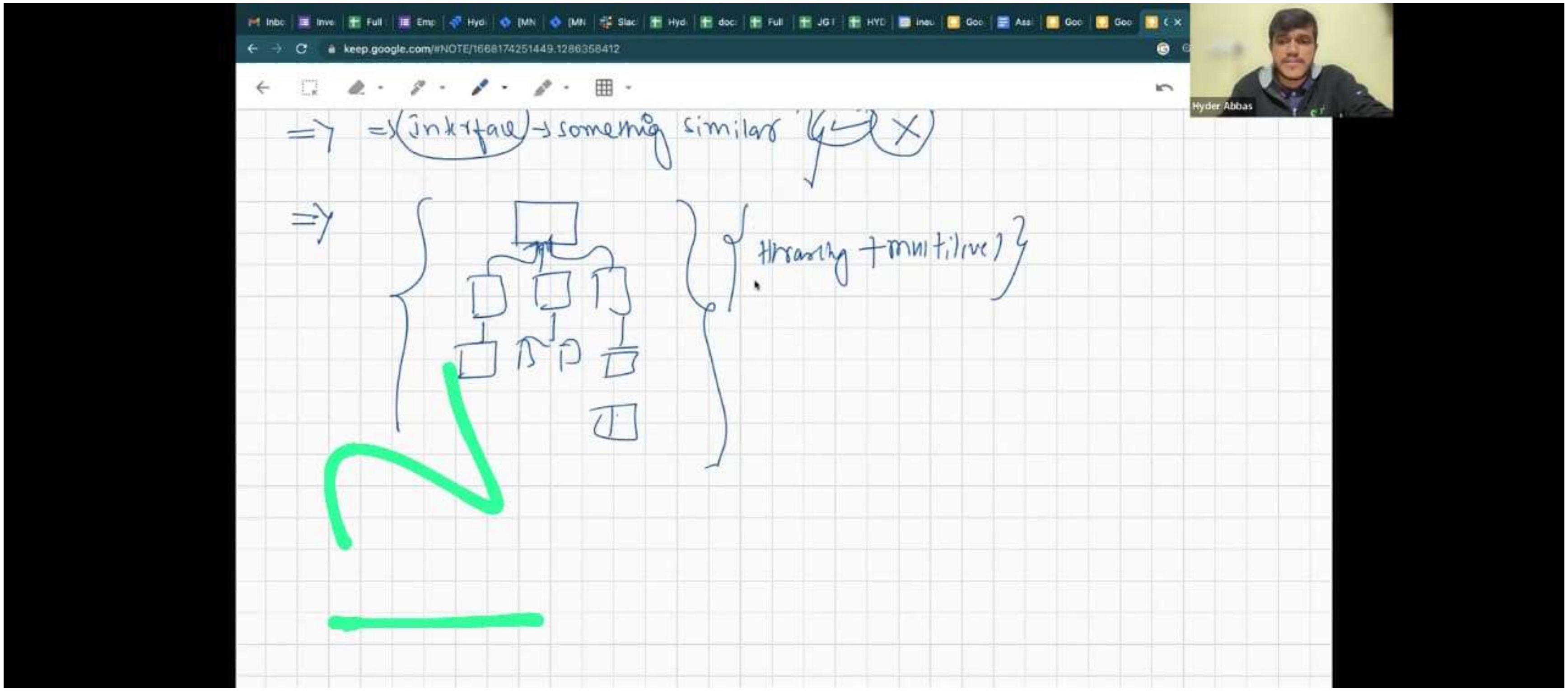
{**}
6

=> multiple Inheritance is not allowed

ambiguous -> compile

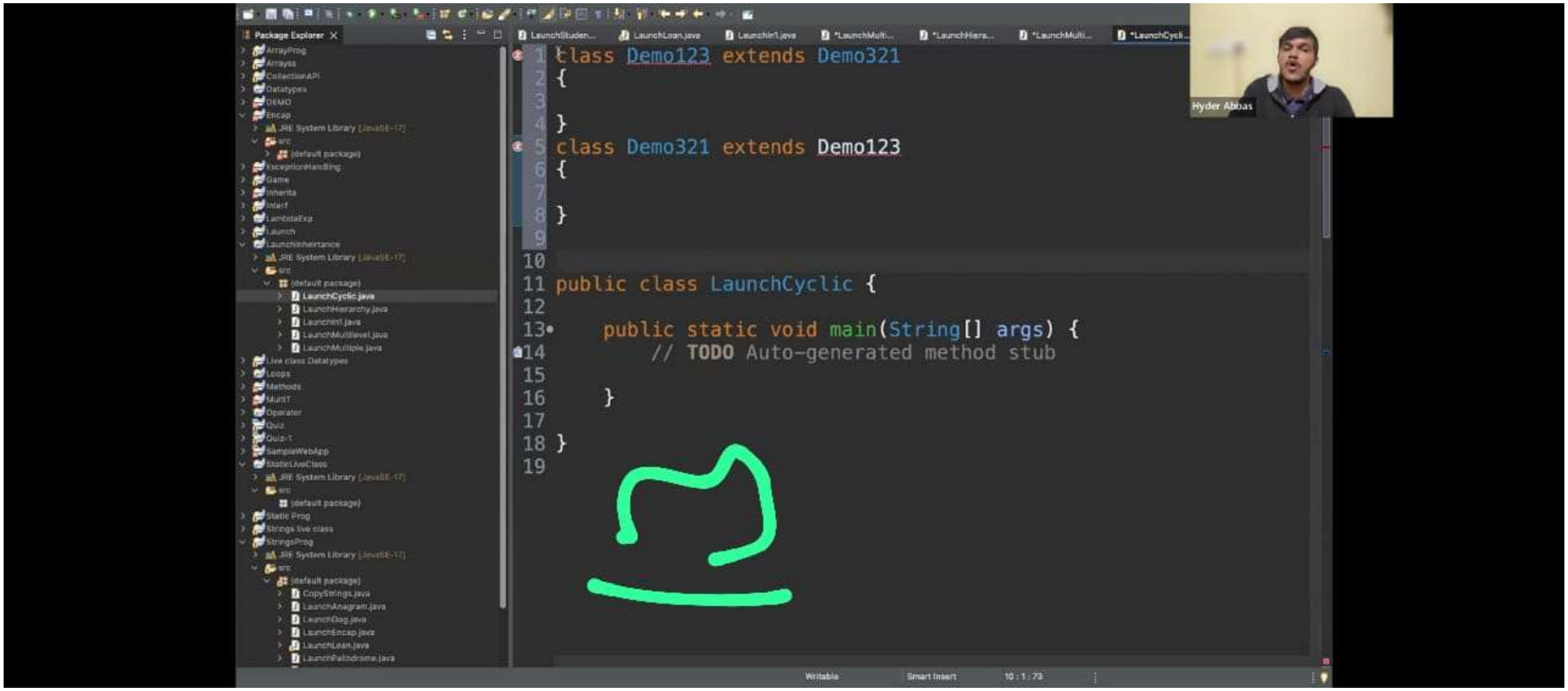


```java
class Parent1
{

}
class Parent2
{


}

//class Child extends Parent1 ,extends Parent2
//{

//}

public class LaunchMultiple {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

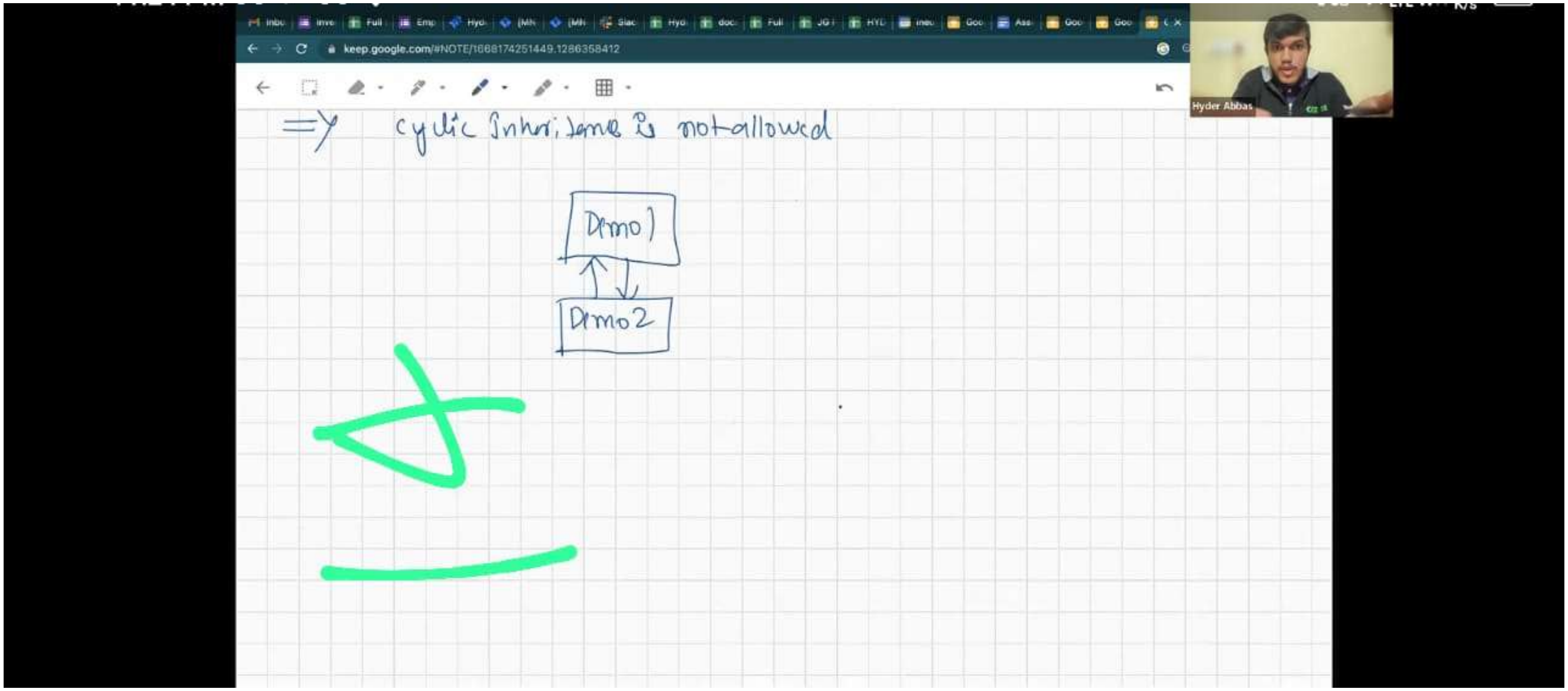


=> =>(interface) -> something similar (X)

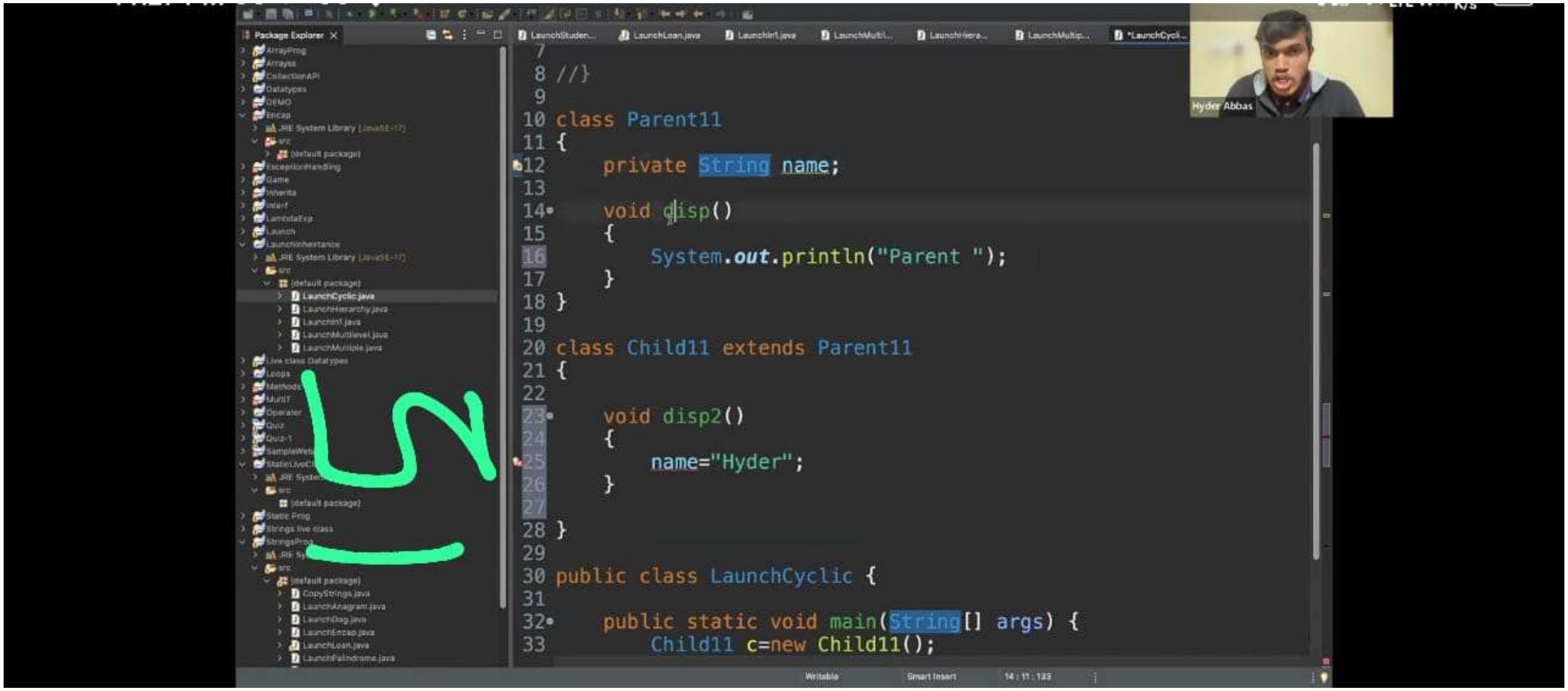
=> { Hirarchy + multilevel }



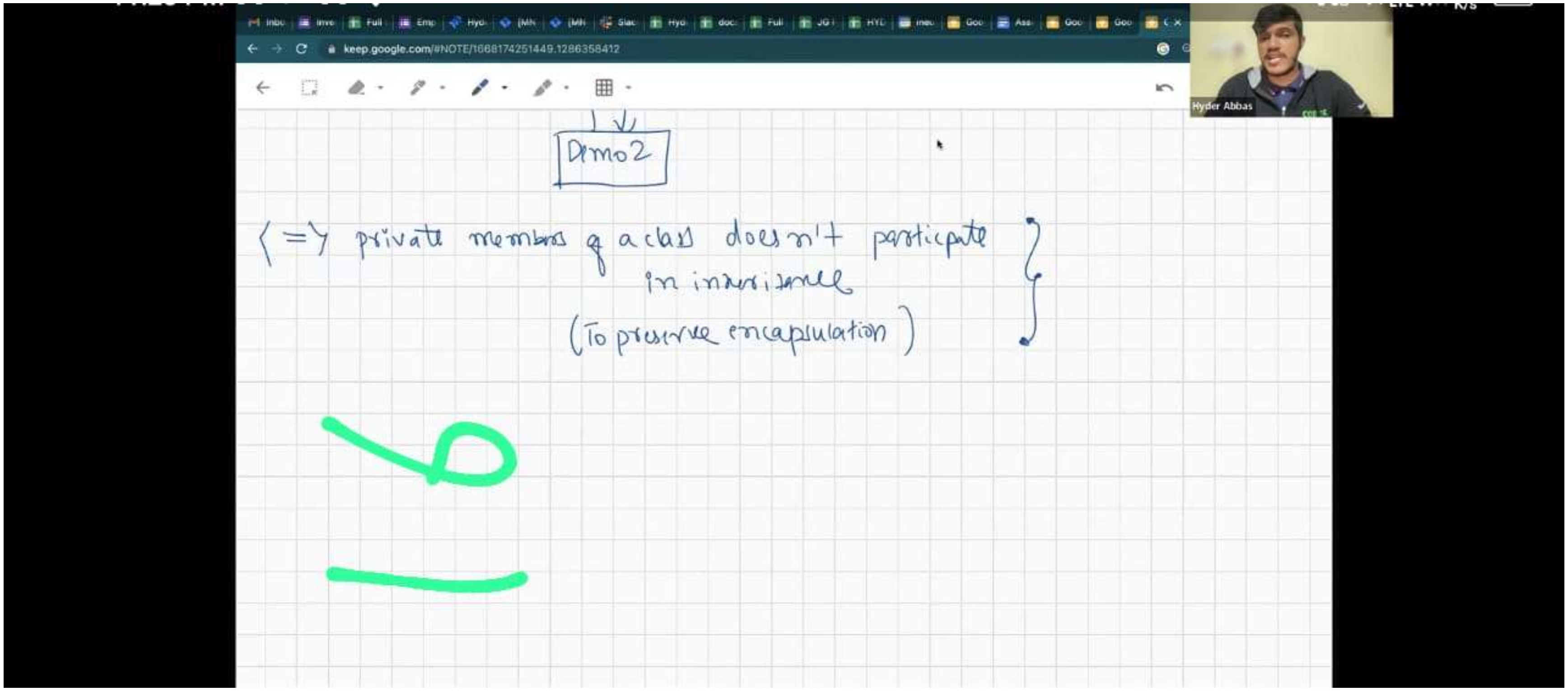
```
class Demo123 extends Demo321
{

}
class Demo321 extends Demo123
{

}

public class LaunchCyclic {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
}
```



=> cyclic Inheritance is not allowed

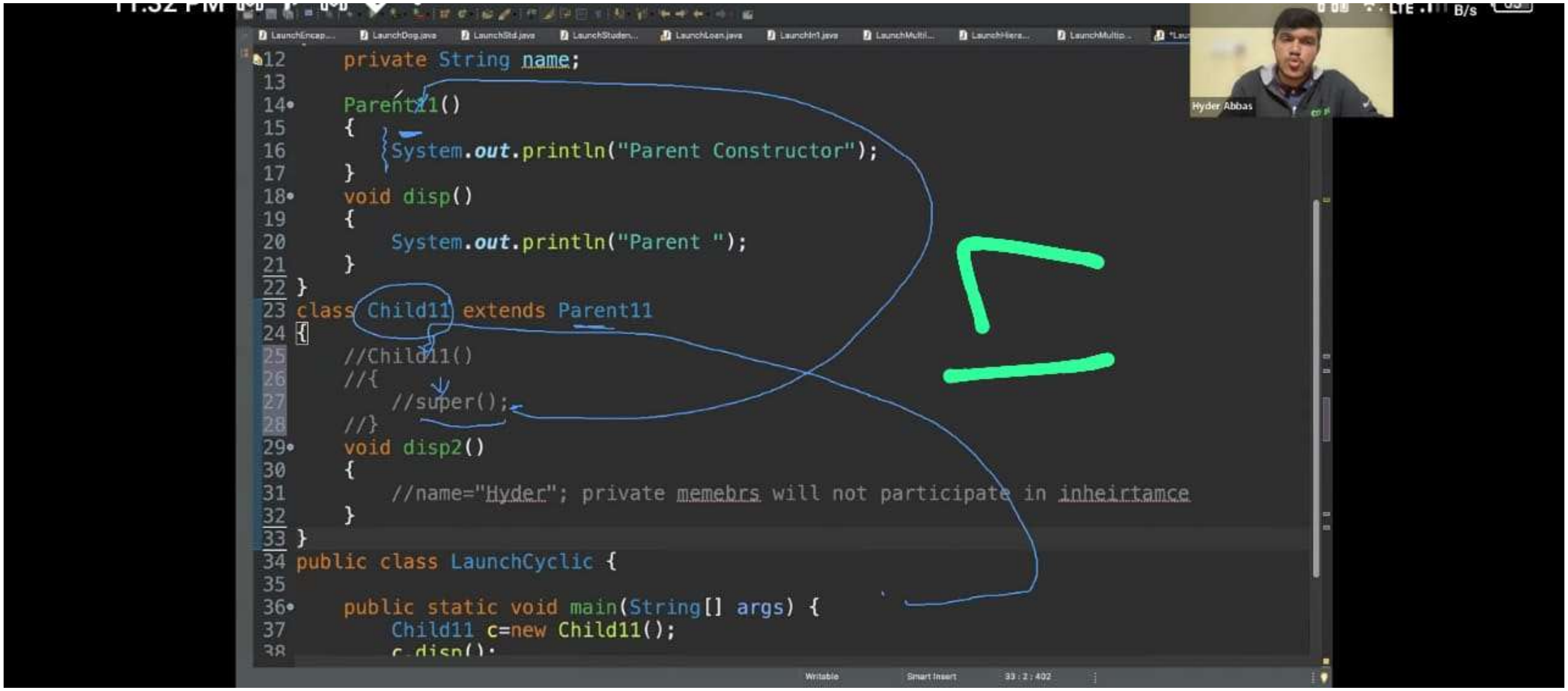


```java
//}

class Parent11
{
    private String name;

    void disp()
    {
        System.out.println("Parent ");
    }
}

class Child11 extends Parent11
{

    void disp2()
    {
        name="Hyder";
    }

}

public class LaunchCyclic {

    public static void main(String[] args) {
        Child11 c=new Child11();
```

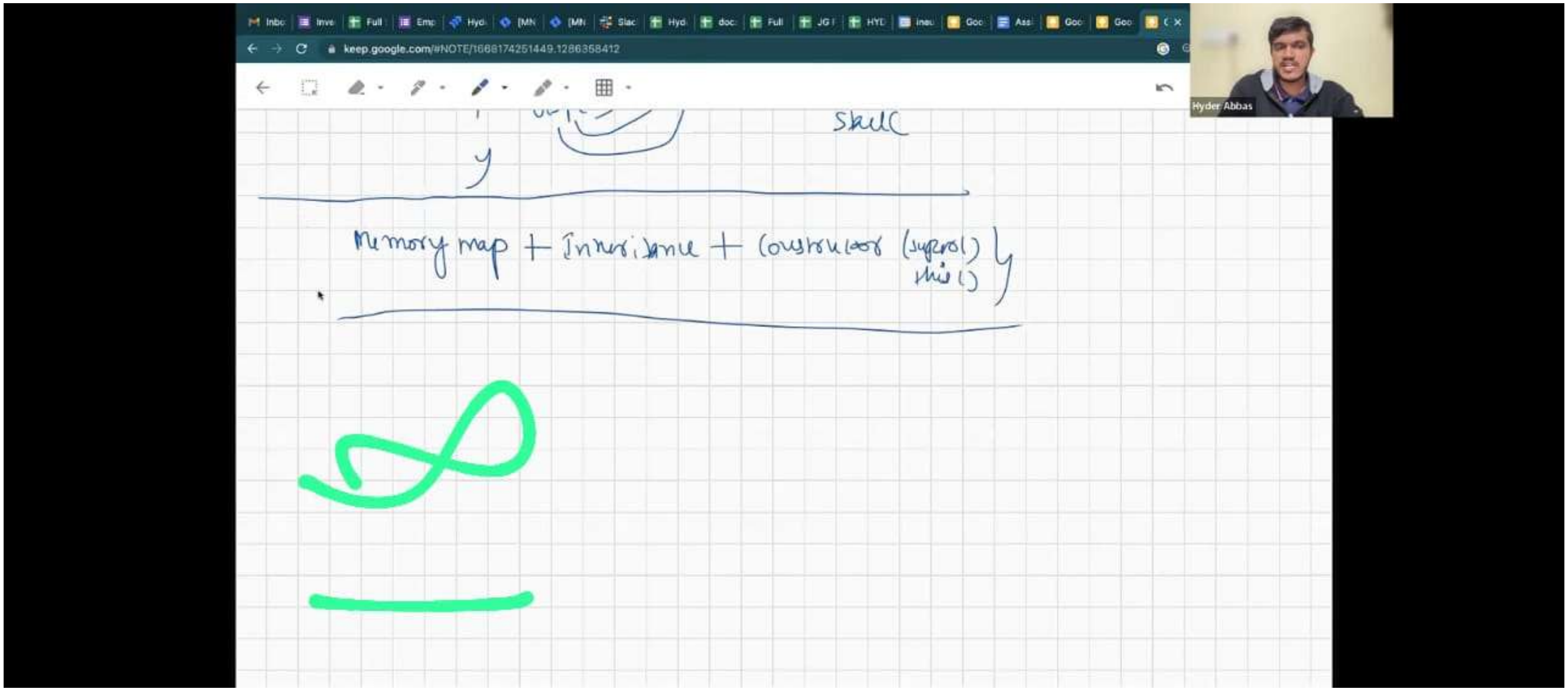


Demo2

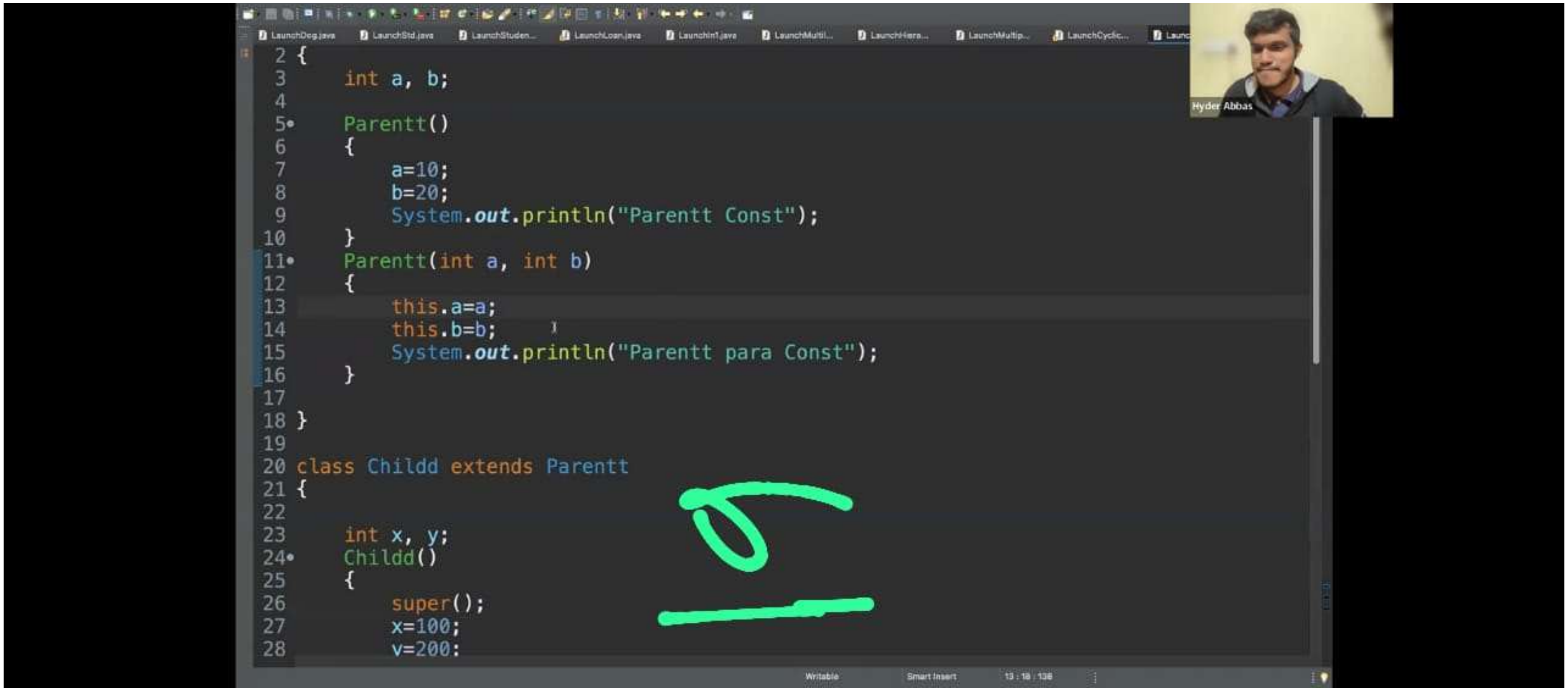
⟨=⟩ private members of a class doesn't participate in inheritance (To preserve encapsulation) }



```
    private String name;

    Parent11()
    {
        System.out.println("Parent Constructor");
    }
    void disp()
    {
        System.out.println("Parent ");
    }
}
class Child11 extends Parent11
{
    //Child11()
    //{
        //super();
    //}
    void disp2()
    {
        //name="Hyder"; private memebrs will not participate in inheirtamce
    }
}
public class LaunchCyclic {

    public static void main(String[] args) {
        Child11 c=new Child11();
        c.disp();
```

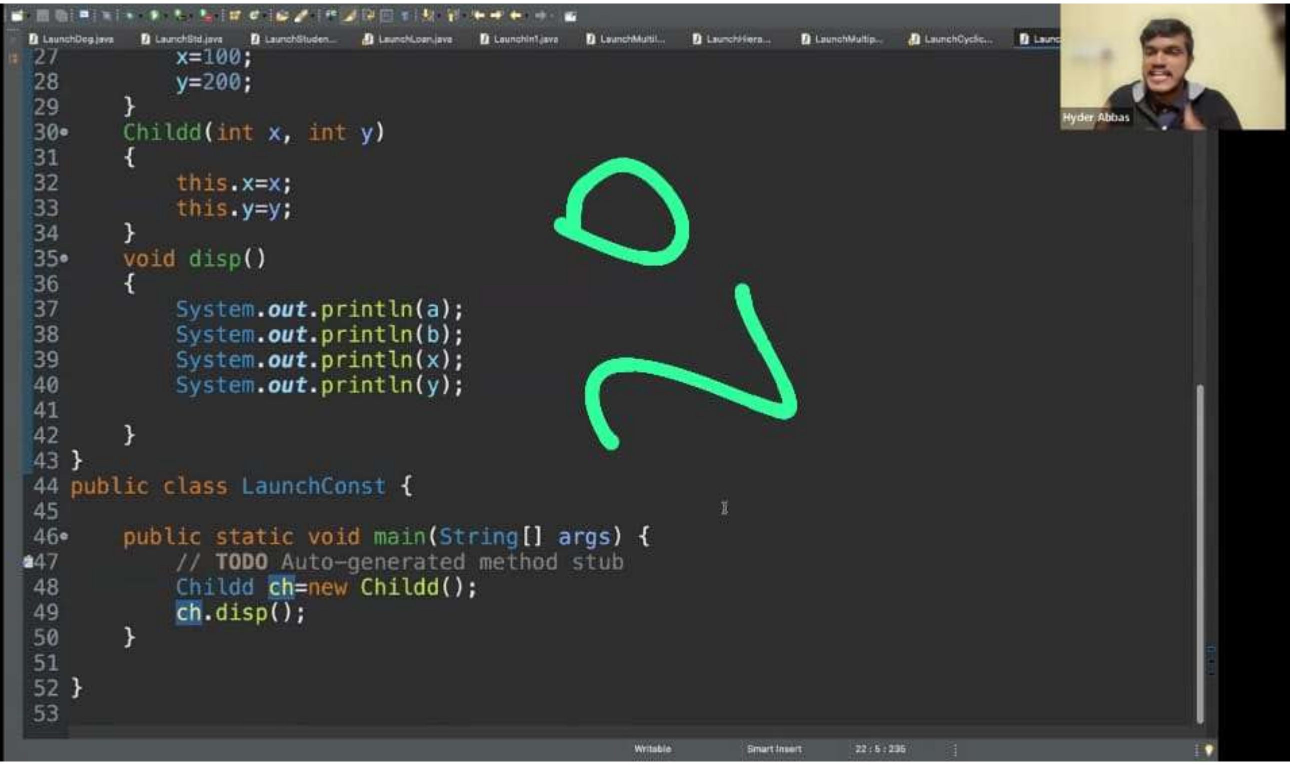


Memory map + Inheritance + Constructor (super()
this())

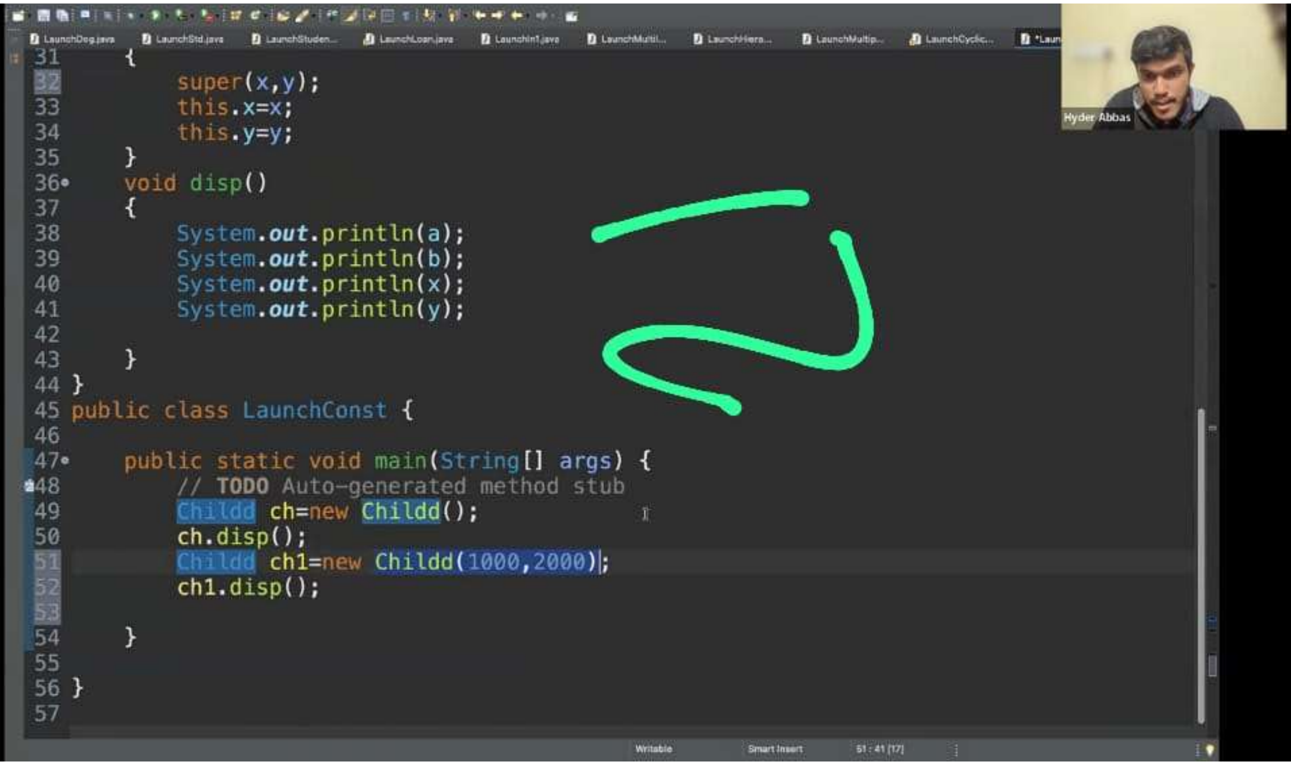


```java
{
    int a, b;

    Parentt()
    {
        a=10;
        b=20;
        System.out.println("Parentt Const");
    }
    Parentt(int a, int b)
    {
        this.a=a;
        this.b=b;
        System.out.println("Parentt para Const");
    }

}

class Childd extends Parentt
{

    int x, y;
    Childd()
    {
        super();
        x=100;
        y=200;
```

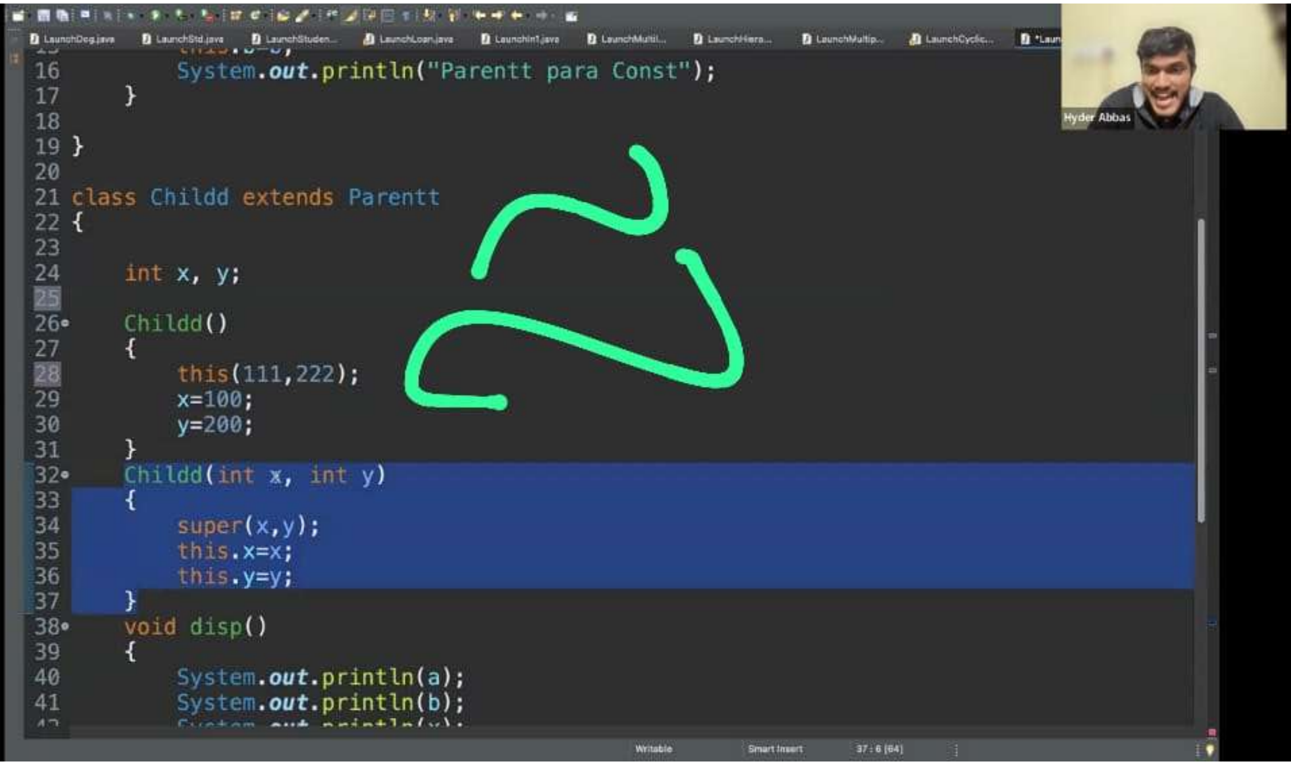
19



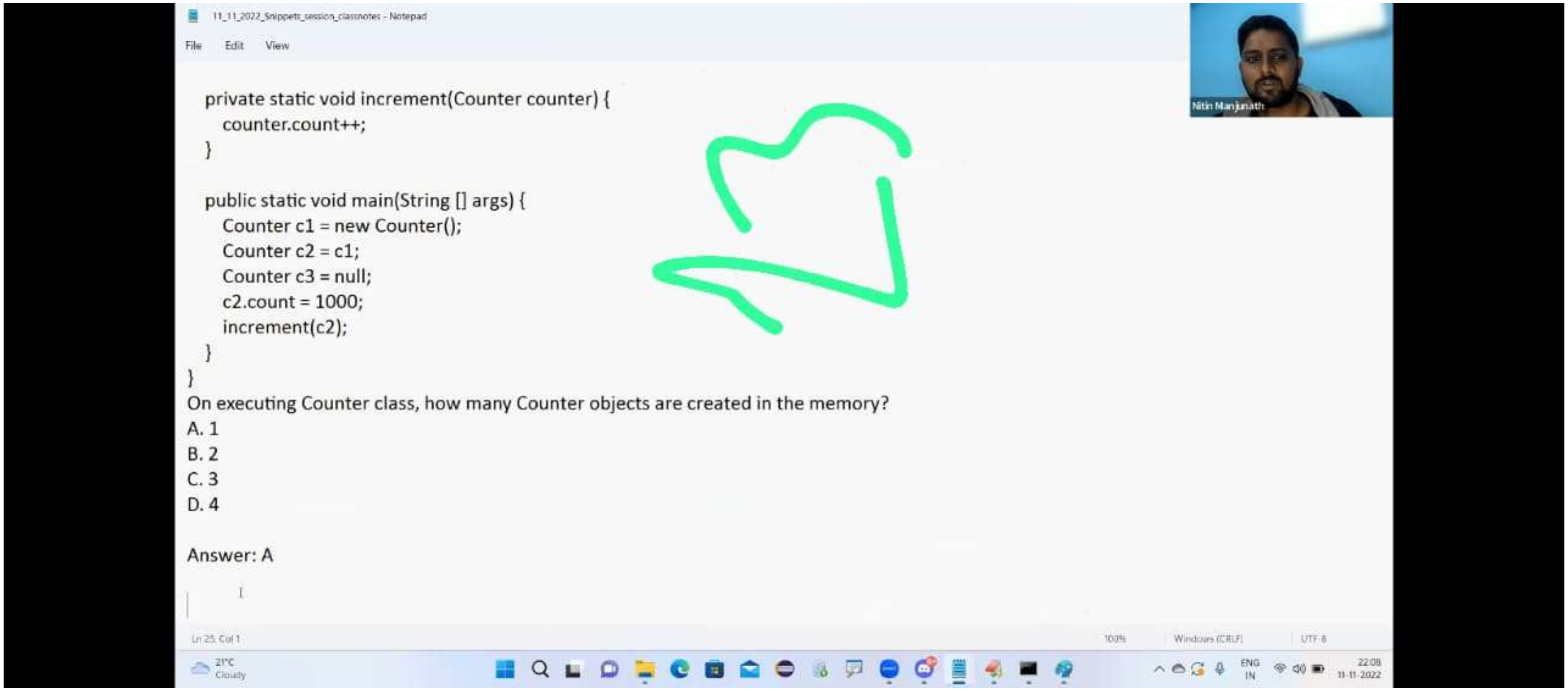
```
        x=100;
        y=200;
    }
    Childd(int x, int y)
    {
        this.x=x;
        this.y=y;
    }
    void disp()
    {
        System.out.println(a);
        System.out.println(b);
        System.out.println(x);
        System.out.println(y);

    }
}
public class LaunchConst {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Childd ch=new Childd();
        ch.disp();
    }

}

```



```java
    {
        super(x,y);
        this.x=x;
        this.y=y;
    }
    void disp()
    {
        System.out.println(a);
        System.out.println(b);
        System.out.println(x);
        System.out.println(y);

    }
}
public class LaunchConst {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Childd ch=new Childd();
        ch.disp();
        Childd ch1=new Childd(1000,2000);
        ch1.disp();

    }

}
```



```java
        System.out.println("Parentt para Const");
    }

}

class Childd extends Parentt
{

    int x, y;

    Childd()
    {
        this(111,222);
        x=100;
        y=200;
    }
    Childd(int x, int y)
    {
        super(x,y);
        this.x=x;
        this.y=y;
    }
    void disp()
    {
        System.out.println(a);
        System.out.println(b);
```



```
  private static void increment(Counter counter) {
    counter.count++;
  }

  public static void main(String [] args) {
    Counter c1 = new Counter();
    Counter c2 = c1;
    Counter c3 = null;
    c2.count = 1000;
    increment(c2);
  }
}
```

On executing Counter class, how many Counter objects are created in the memory?

A. 1

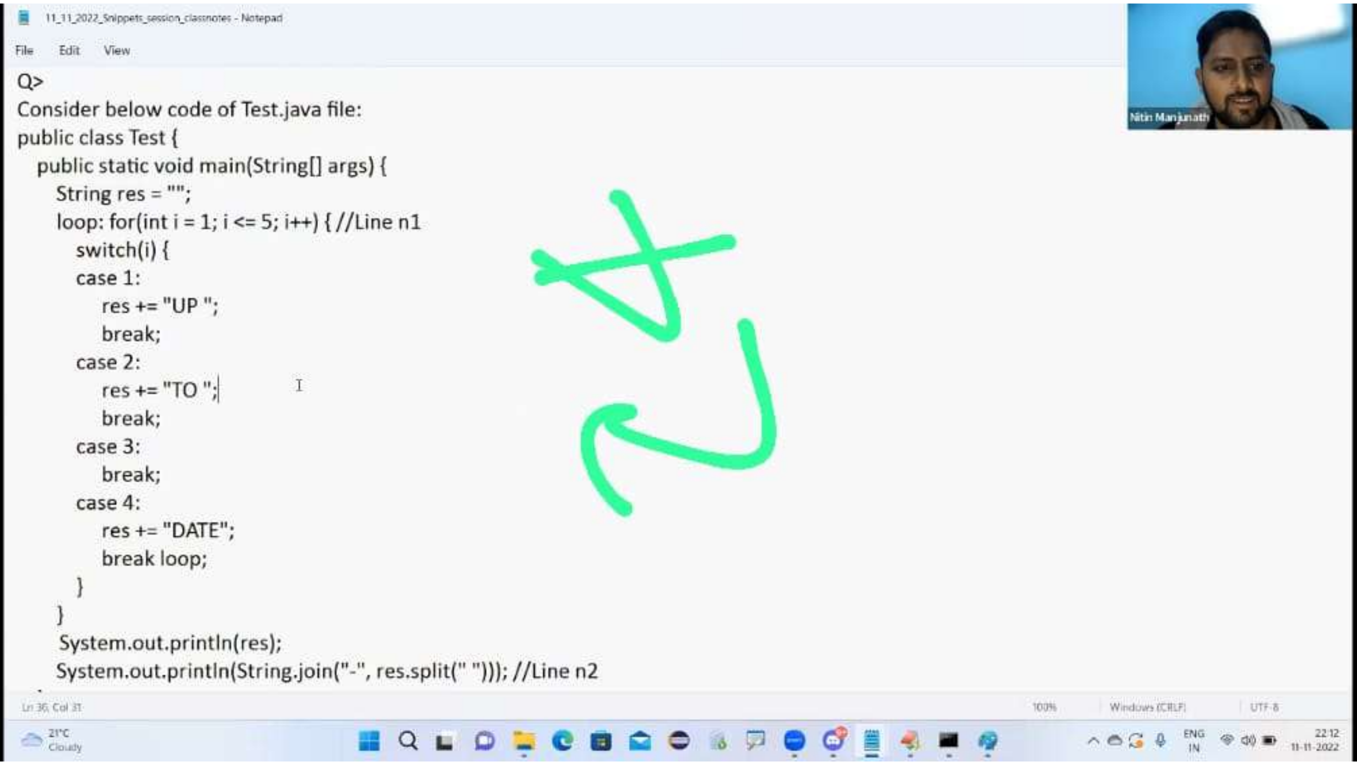
B. 2

C. 3

D. 4

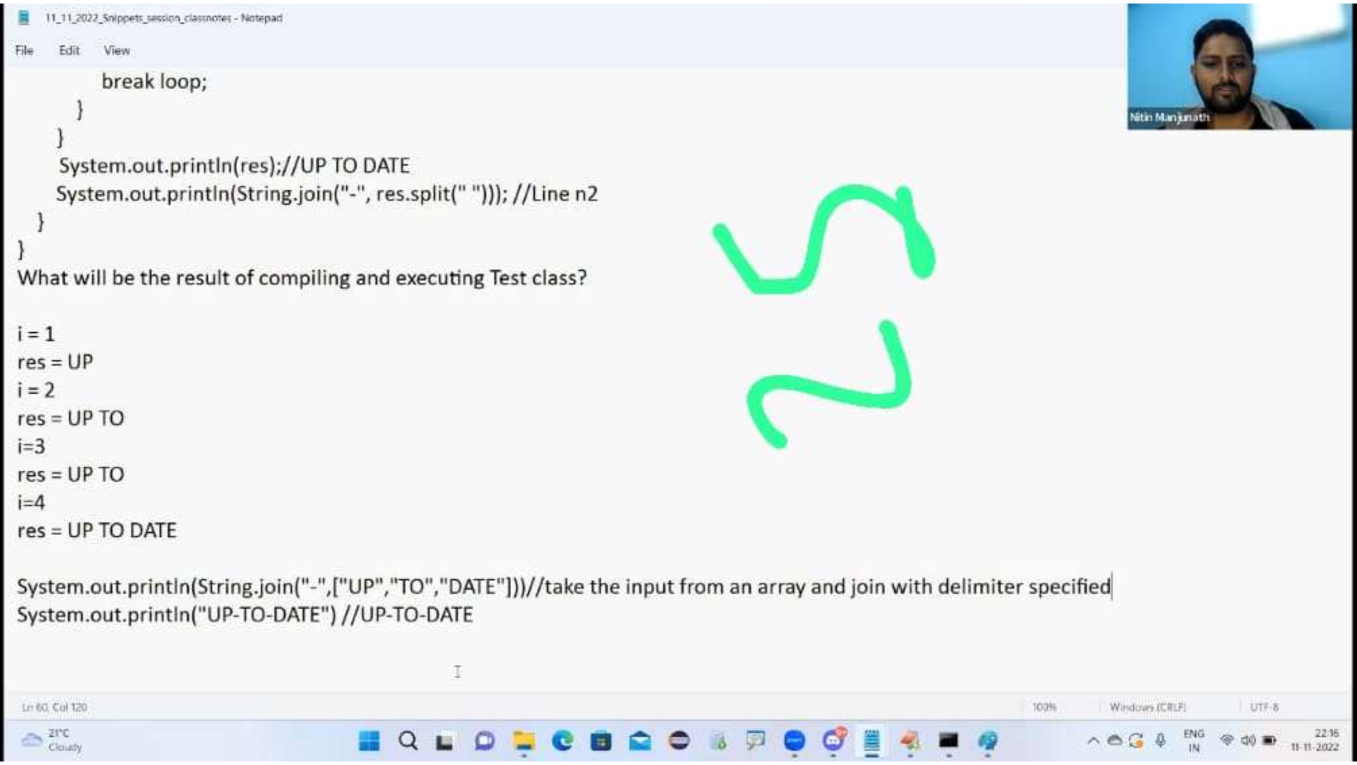
Answer: A

23



Q>
Consider below code of Test.java file:

```
public class Test {
  public static void main(String[] args) {
    String res = "";
    loop: for(int i = 1; i <= 5; i++) { //Line n1
      switch(i) {
      case 1:
         res += "UP ";
         break;
      case 2:
         res += "TO ";
         break;
      case 3:
         break;
      case 4:
         res += "DATE";
         break loop;
      }
    }
    System.out.println(res);
    System.out.println(String.join("-", res.split(" "))); //Line n2
```

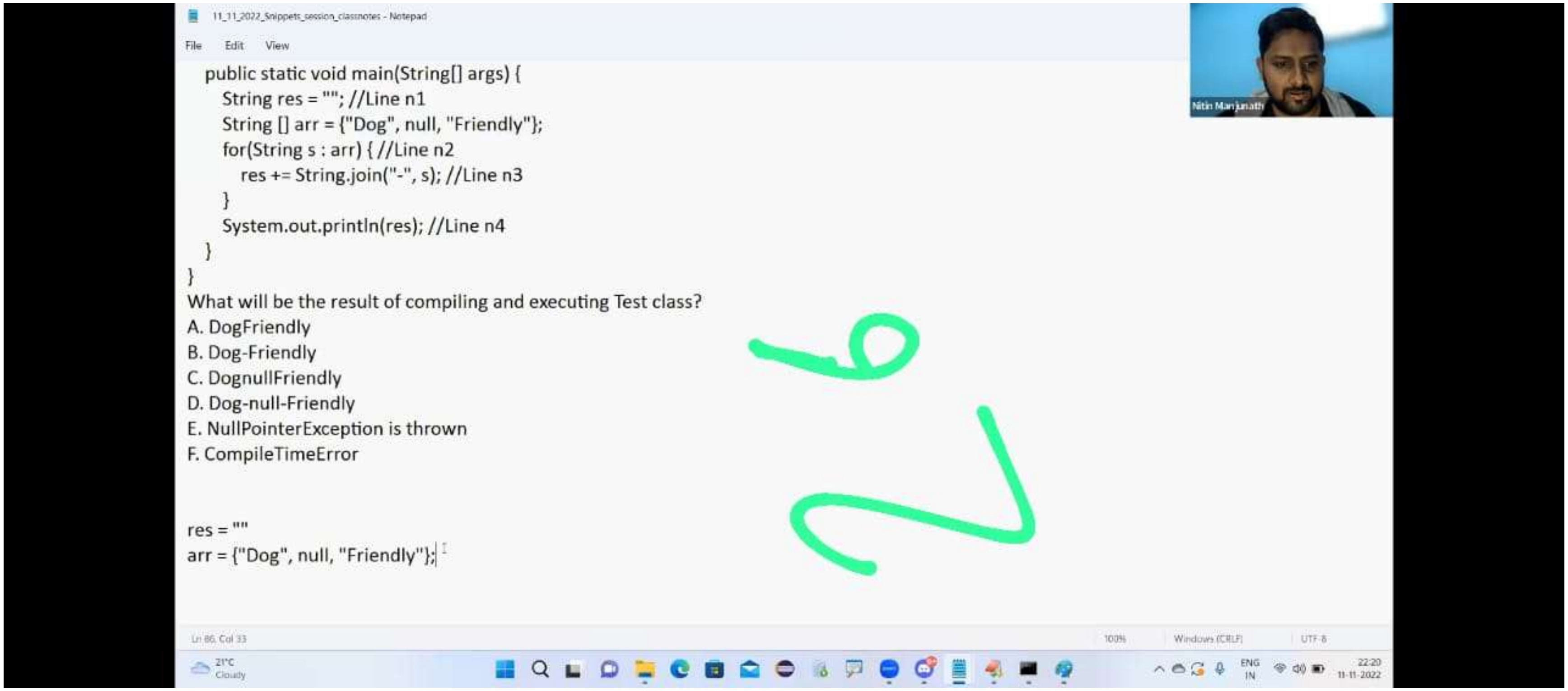


```
            break loop;
          }
        }
        System.out.println(res);//UP TO DATE
       System.out.println(String.join("-", res.split(" "))); //Line n2
    }
}
```

What will be the result of compiling and executing Test class?

```
i = 1
res = UP
i = 2
res = UP TO
i=3
res = UP TO
i=4
res = UP TO DATE

System.out.println(String.join("-",["UP","TO","DATE"]))//take the input from an array and join with delimiter specified
System.out.println("UP-TO-DATE") //UP-TO-DATE
```

25



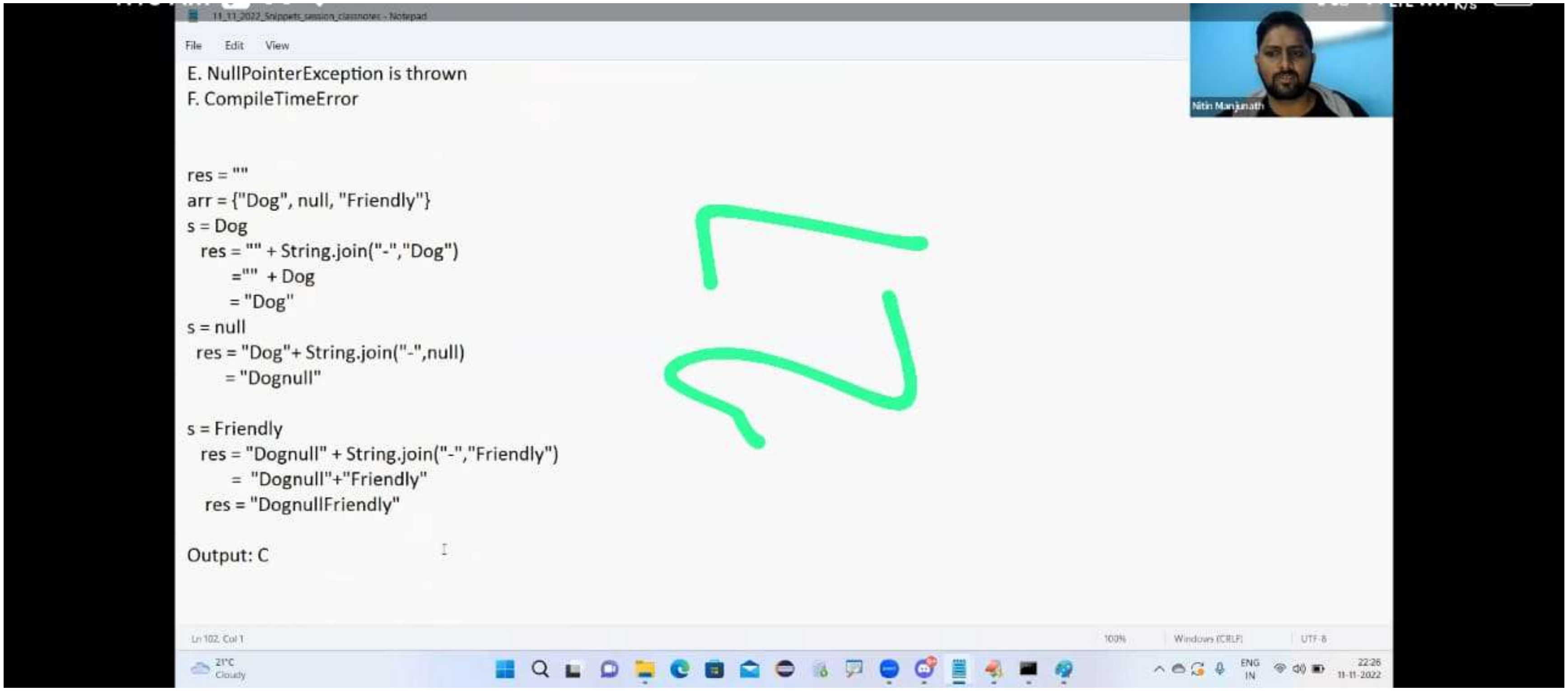
```
  public static void main(String[] args) {
    String res = ""; //Line n1
    String [] arr = {"Dog", null, "Friendly"};
    for(String s : arr) { //Line n2
      res += String.join("-", s); //Line n3
    }
    System.out.println(res); //Line n4
  }
}
```

What will be the result of compiling and executing Test class?

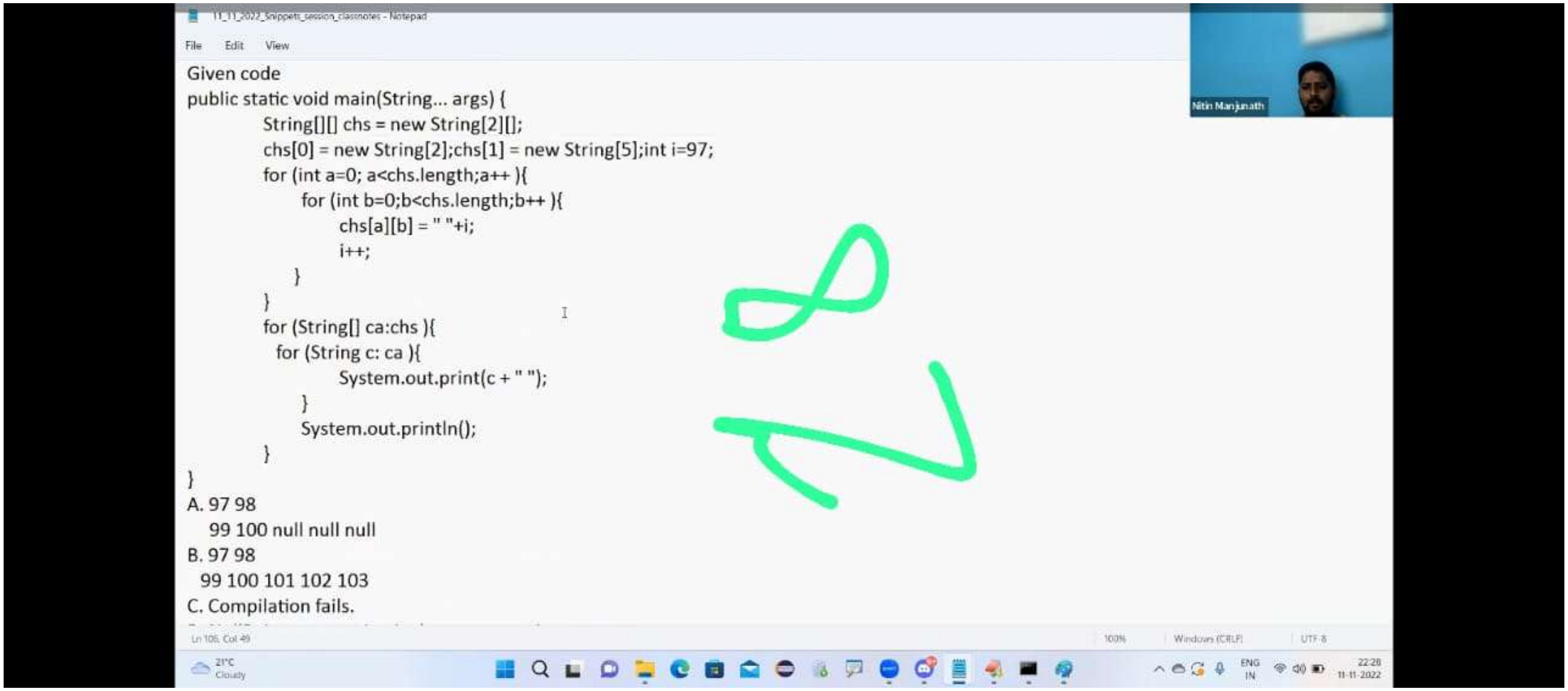
A. DogFriendly
B. Dog-Friendly
C. DognullFriendly
D. Dog-null-Friendly
E. NullPointerException is thrown
F. CompileTimeError

```
res = ""
arr = {"Dog", null, "Friendly"};
```

26



```
E. NullPointerException is thrown
F. CompileTimeError


res = ""
arr = {"Dog", null, "Friendly"}
s = Dog
  res = "" + String.join("-","Dog")
       =""  + Dog
       = "Dog"
s = null
 res = "Dog"+ String.join("-",null)
      = "Dognull"

s = Friendly
  res = "Dognull" + String.join("-","Friendly")
       =  "Dognull"+"Friendly"
   res = "DognullFriendly"

Output: C
```



Given code

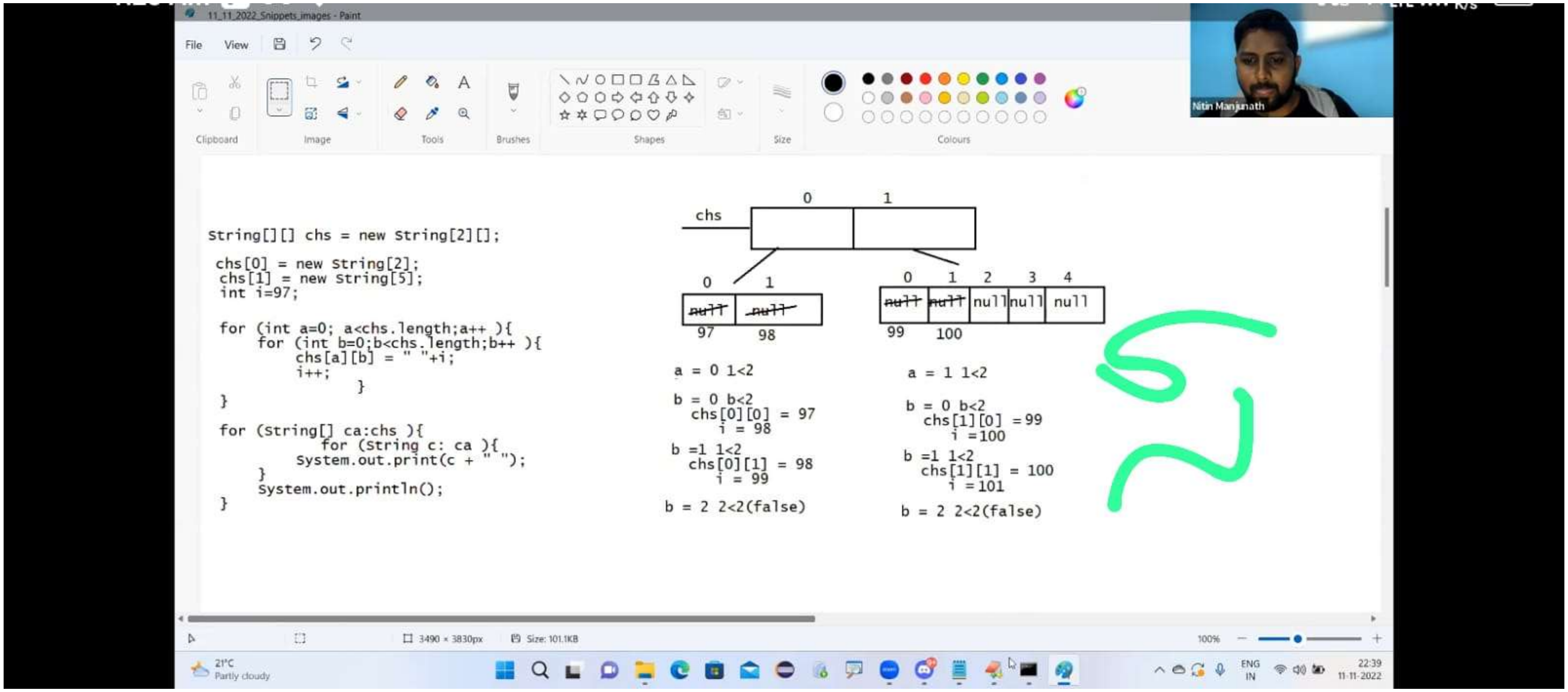
```
public static void main(String... args) {
            String[][] chs = new String[2][];
            chs[0] = new String[2];chs[1] = new String[5];int i=97;
            for (int a=0; a<chs.length;a++ ){
                  for (int b=0;b<chs.length;b++ ){
                        chs[a][b] = " "+i;
                        i++;
                  }
            }
            for (String[] ca:chs ){
              for (String c: ca ){
                        System.out.print(c + " ");
                  }
                  System.out.println();
            }
}
```

A. 97 98
   99 100 null null null

B. 97 98
  99 100 101 102 103

C. Compilation fails.

28



```
String[][] chs = new String[2][];

 chs[0] = new String[2];
 chs[1] = new String[5];
 int i=97;

 for (int a=0; a<chs.length;a++ ){
     for (int b=0;b<chs.length;b++ ){
         chs[a][b] = " "+i;
         i++;
               }
 }

 for (String[] ca:chs ){
          for (String c: ca ){
         System.out.print(c + " ");
     }
     System.out.println();
 }
```

a = 0 1<2

b = 0 b<2
 chs[0][0] = 97
 i = 98

b =1 1<2
 chs[0][1] = 98
 i = 99

b = 2 2<2(false)

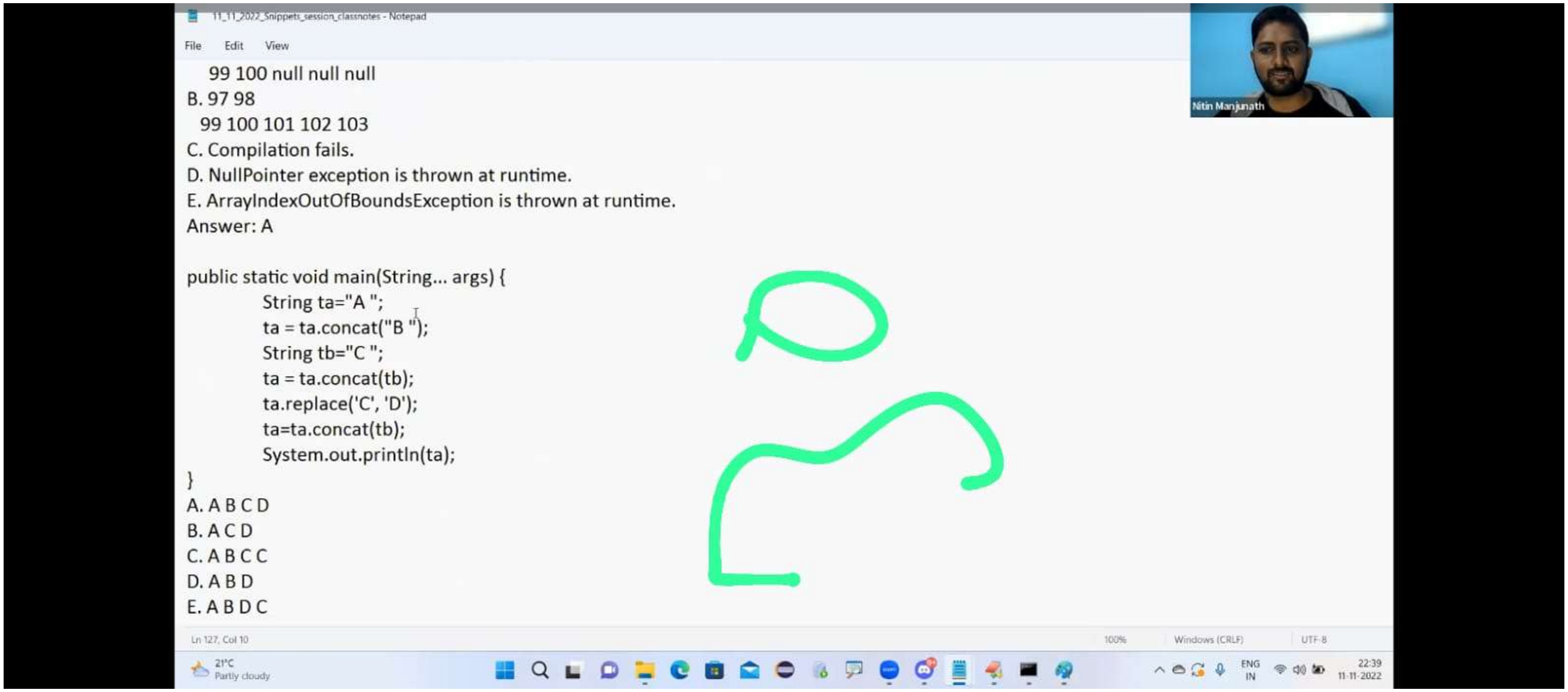
a = 1 1<2

b = 0 b<2
 chs[1][0] = 99
 i =100

b =1 1<2
 chs[1][1] = 100
 i =101

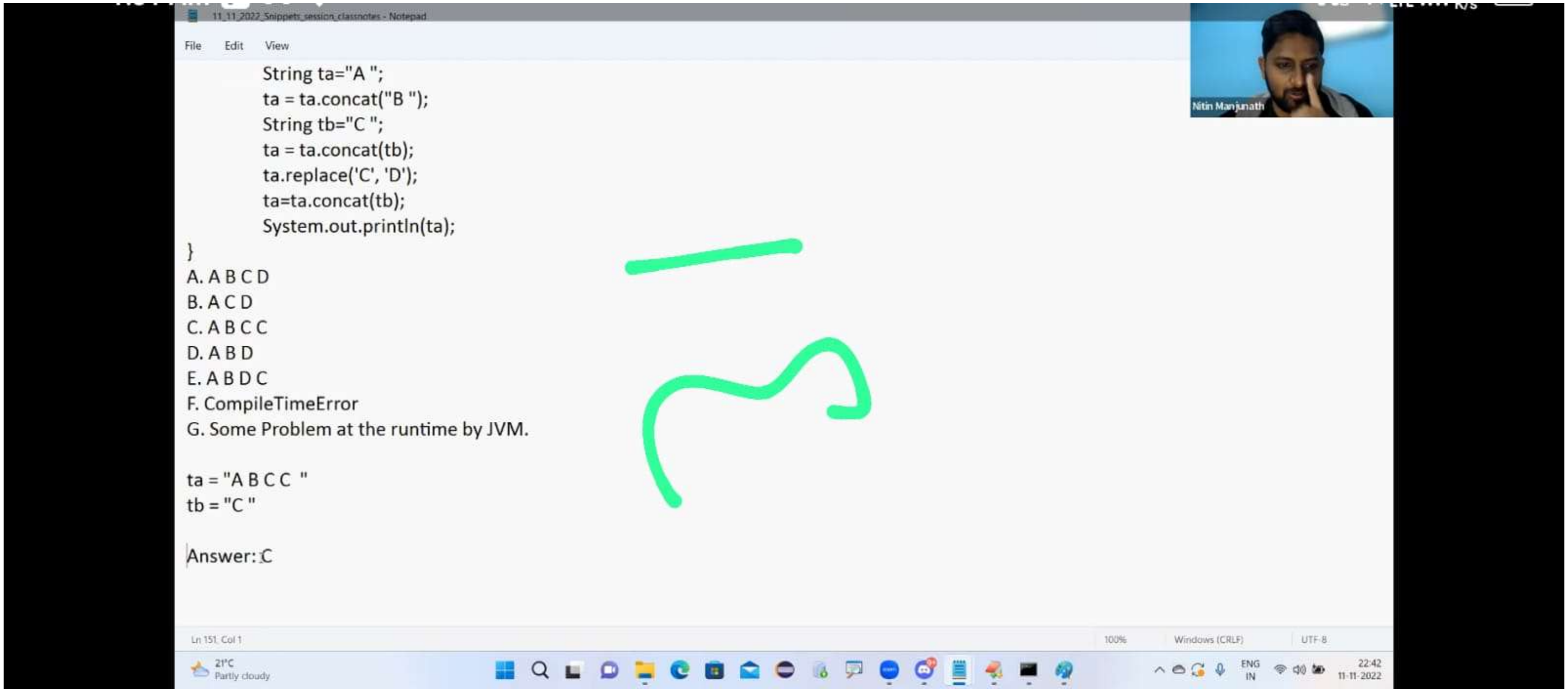
b = 2 2<2(false)

29



```
   99 100 null null null
B. 97 98
  99 100 101 102 103
C. Compilation fails.
D. NullPointer exception is thrown at runtime.
E. ArrayIndexOutOfBoundsException is thrown at runtime.
Answer: A

public static void main(String... args) {
            String ta="A ";
            ta = ta.concat("B ");
            String tb="C ";
            ta = ta.concat(tb);
            ta.replace('C', 'D');
            ta=ta.concat(tb);
            System.out.println(ta);
}
A. A B C D
B. A C D
C. A B C C
D. A B D
E. A B D C
```

30



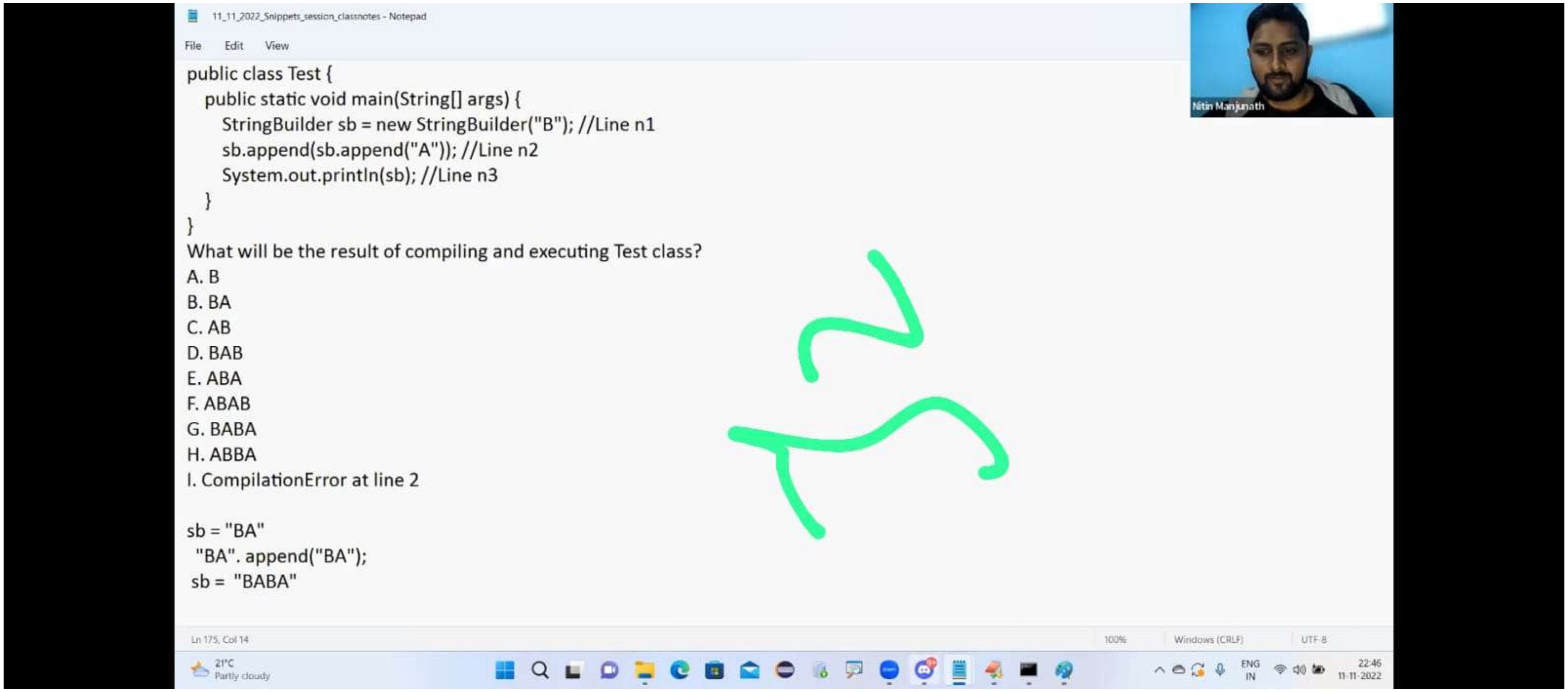
```
            String ta="A ";
            ta = ta.concat("B ");
            String tb="C ";
            ta = ta.concat(tb);
            ta.replace('C', 'D');
            ta=ta.concat(tb);
            System.out.println(ta);
}
```

A. A B C D
B. A C D
C. A B C C
D. A B D
E. A B D C
F. CompileTimeError
G. Some Problem at the runtime by JVM.

ta = "A B C C "
tb = "C "

Answer: C

31

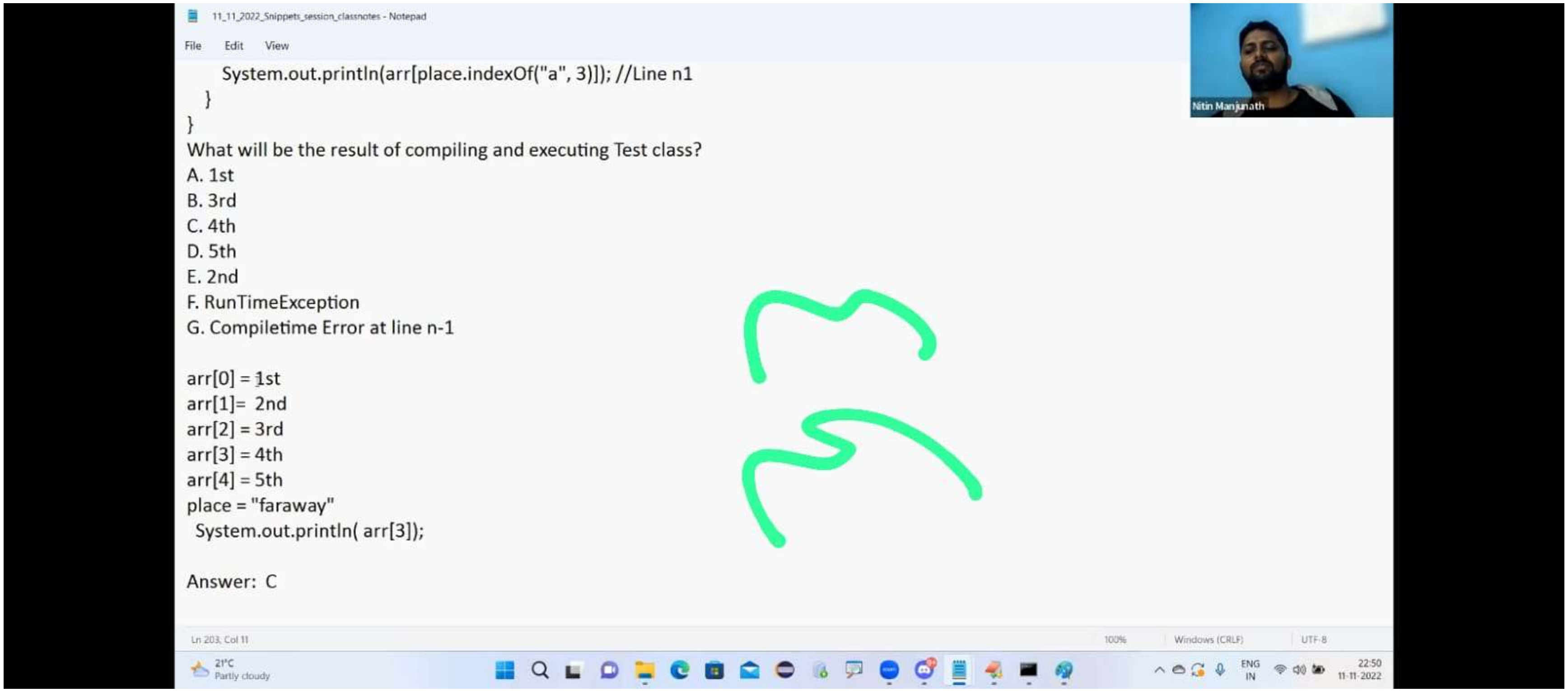


```
public class Test {
  public static void main(String[] args) {
    StringBuilder sb = new StringBuilder("B"); //Line n1
    sb.append(sb.append("A")); //Line n2
    System.out.println(sb); //Line n3
  }
}
```

What will be the result of compiling and executing Test class?

A. B
B. BA
C. AB
D. BAB
E. ABA
F. ABAB
G. BABA
H. ABBA
I. CompilationError at line 2

```
sb = "BA"
 "BA". append("BA");
sb =  "BABA"
```



```
      System.out.println(arr[place.indexOf("a", 3)]); //Line n1
   }
}
```

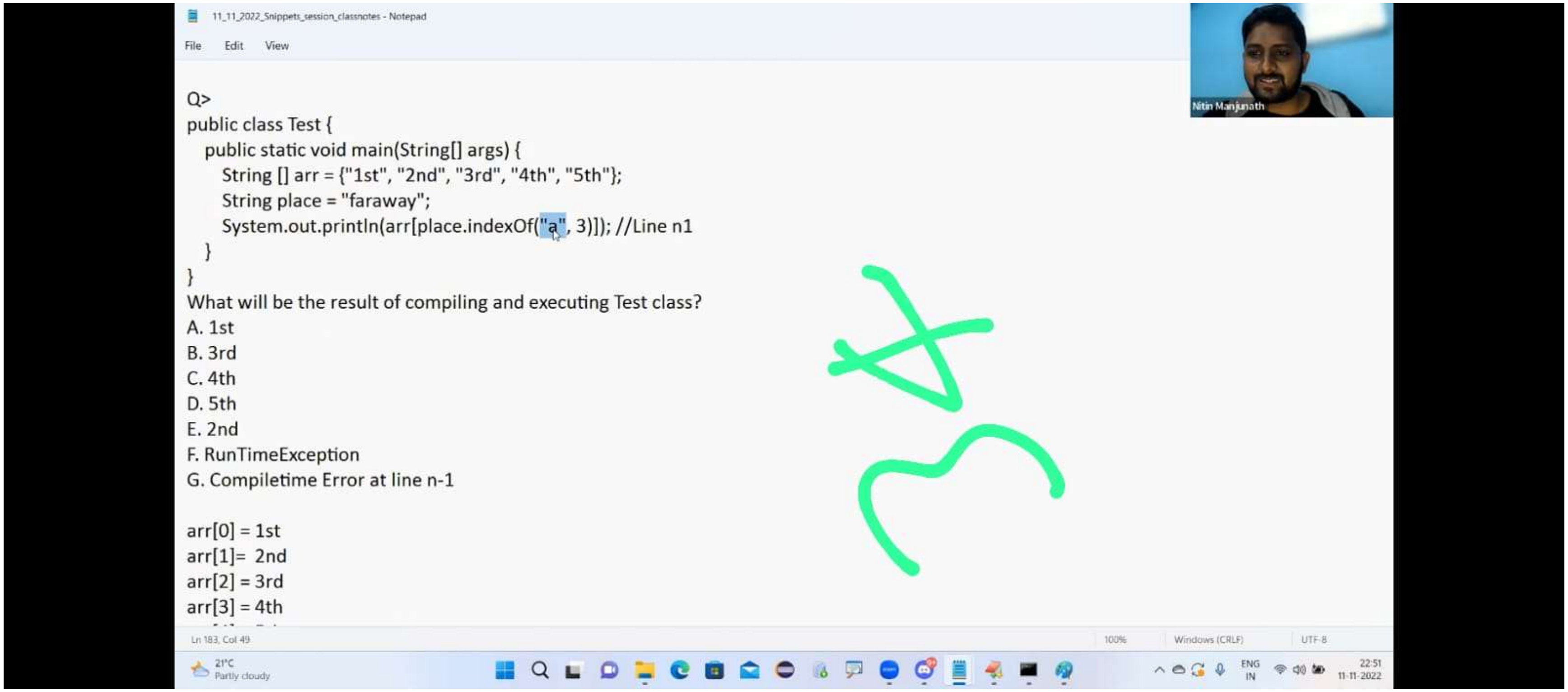
What will be the result of compiling and executing Test class?

A. 1st
B. 3rd
C. 4th
D. 5th
E. 2nd
F. RunTimeException
G. Compiletime Error at line n-1

```
arr[0] = 1st
arr[1]=  2nd
arr[2] = 3rd
arr[3] = 4th
arr[4] = 5th
place = "faraway"
 System.out.println( arr[3]);
```

Answer: C

33



```
Q>
public class Test {
  public static void main(String[] args) {
    String [] arr = {"1st", "2nd", "3rd", "4th", "5th"};
    String place = "faraway";
    System.out.println(arr[place.indexOf("a", 3)]); //Line n1
  }
}
```

What will be the result of compiling and executing Test class?

A. 1st
B. 3rd
C. 4th
D. 5th
E. 2nd
F. RunTimeException
G. Compiletime Error at line n-1

```
arr[0] = 1st
arr[1]=  2nd
arr[2] = 3rd
arr[3] = 4th
```

34