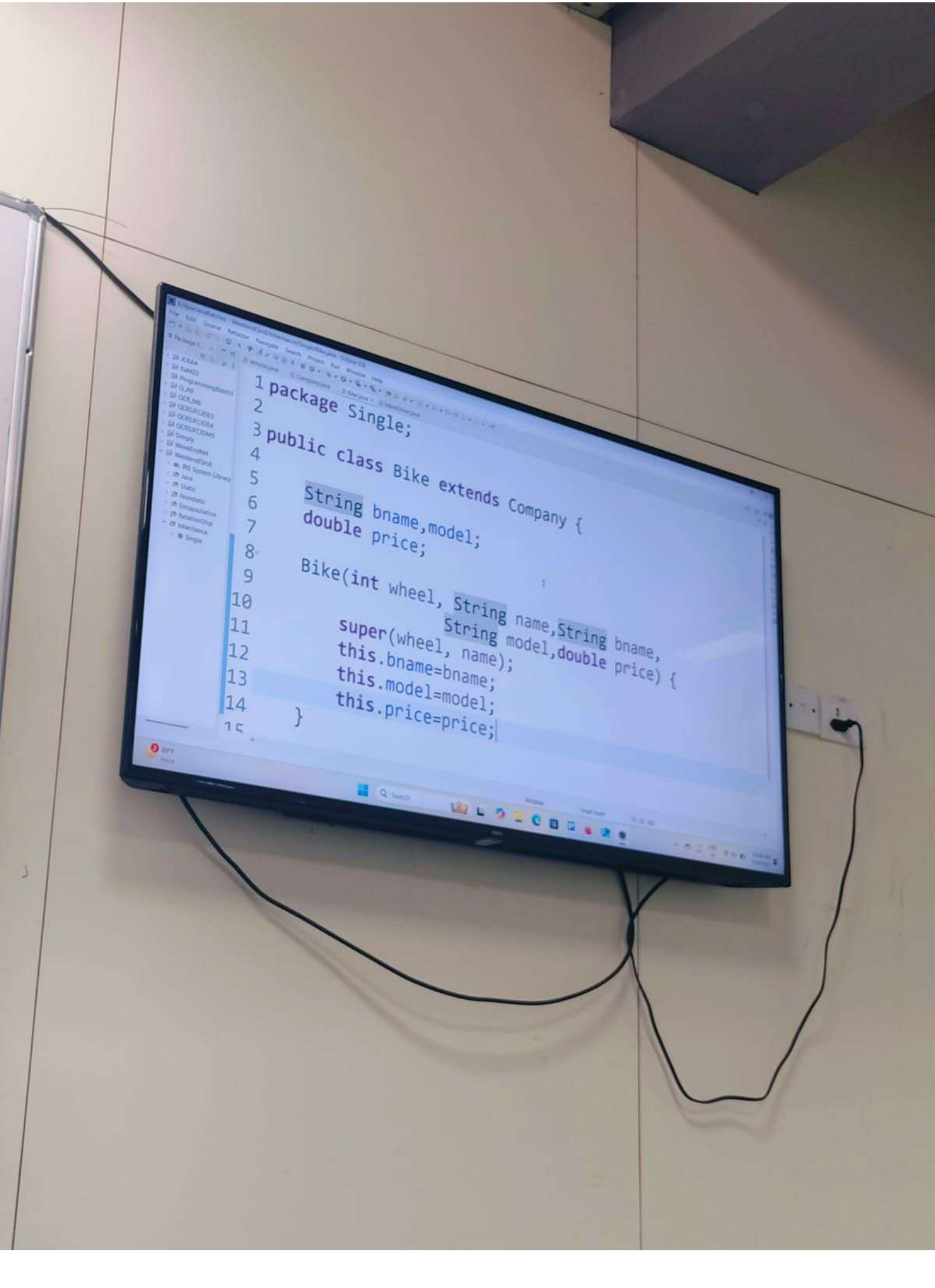
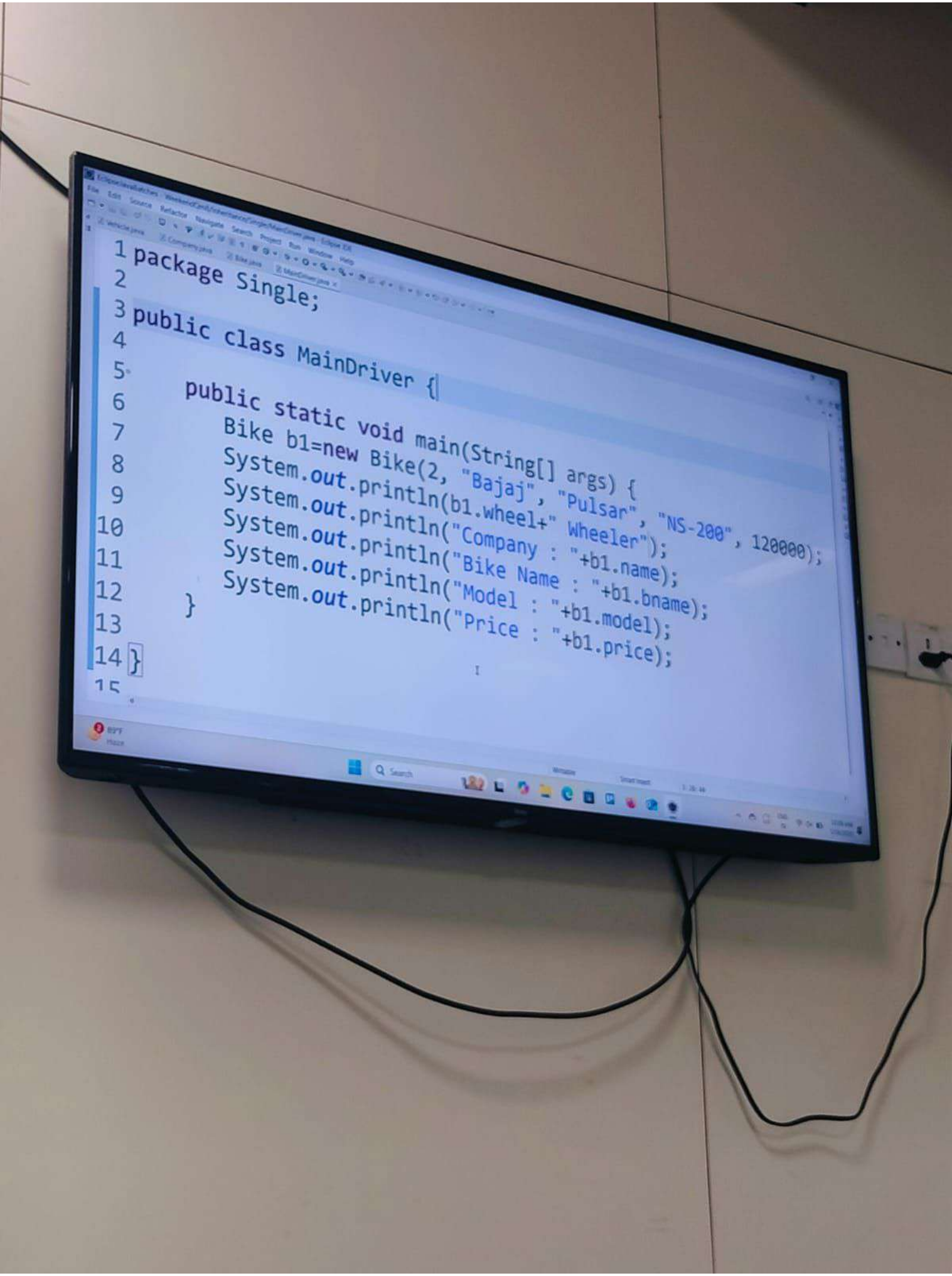
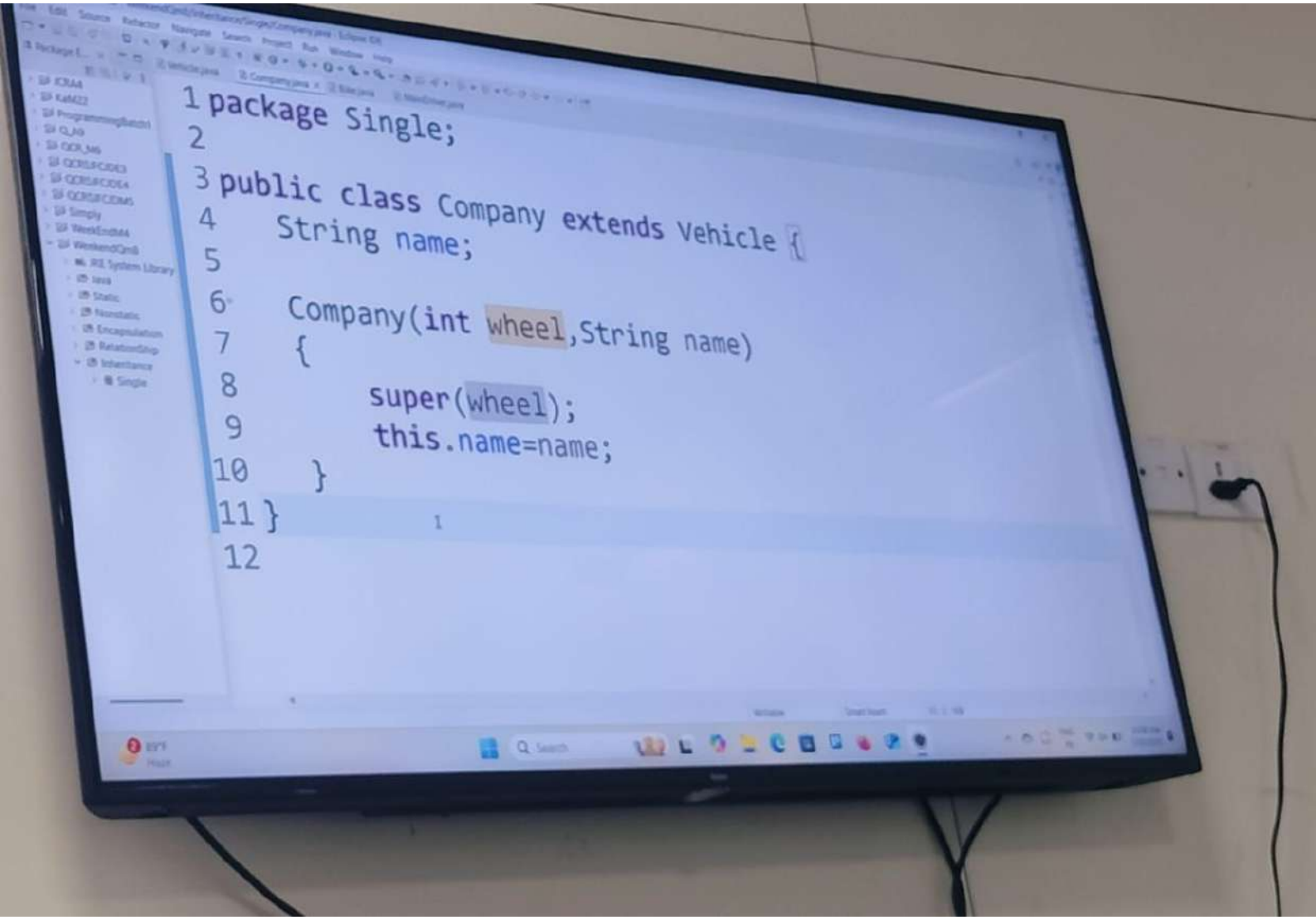
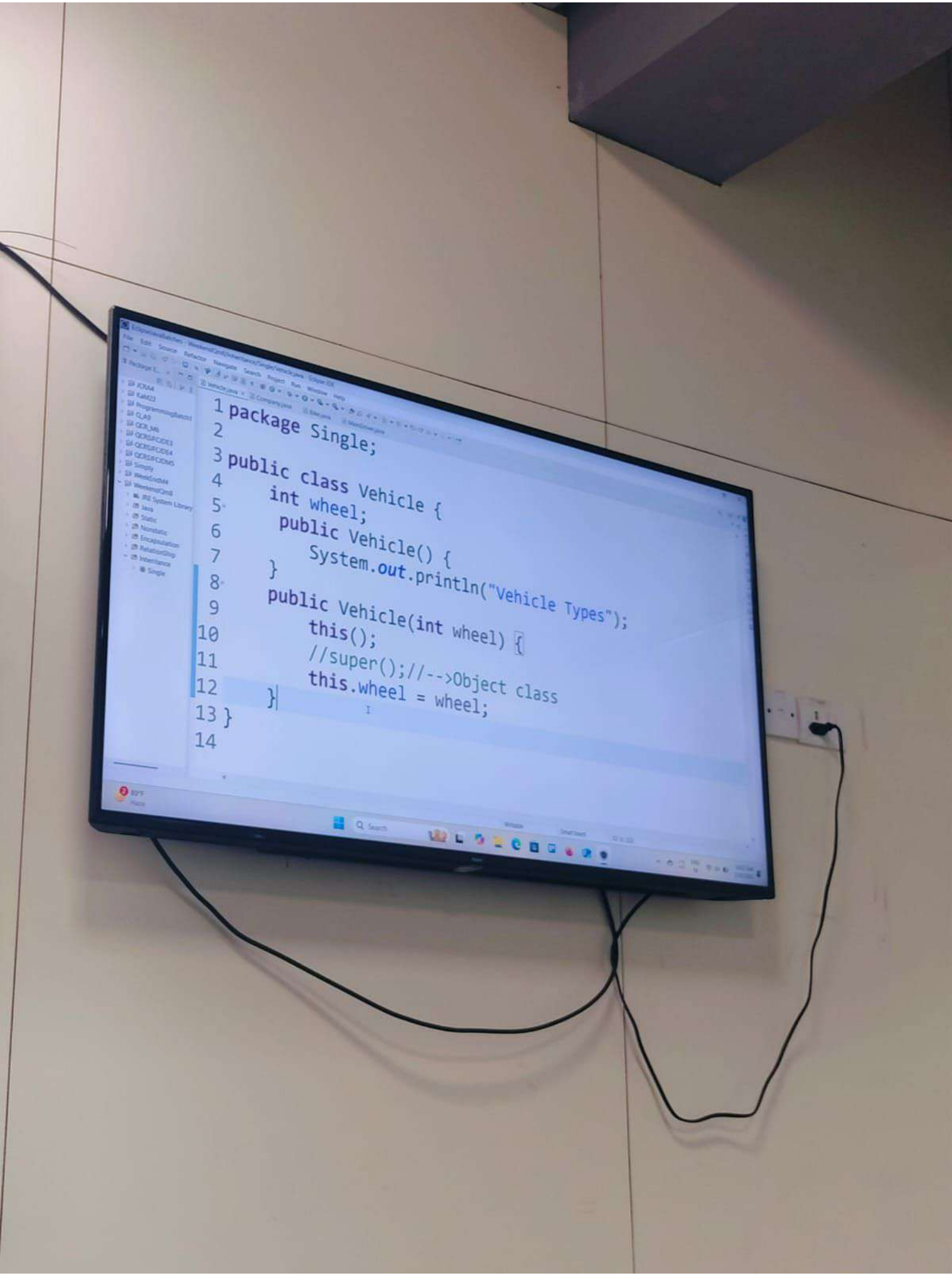
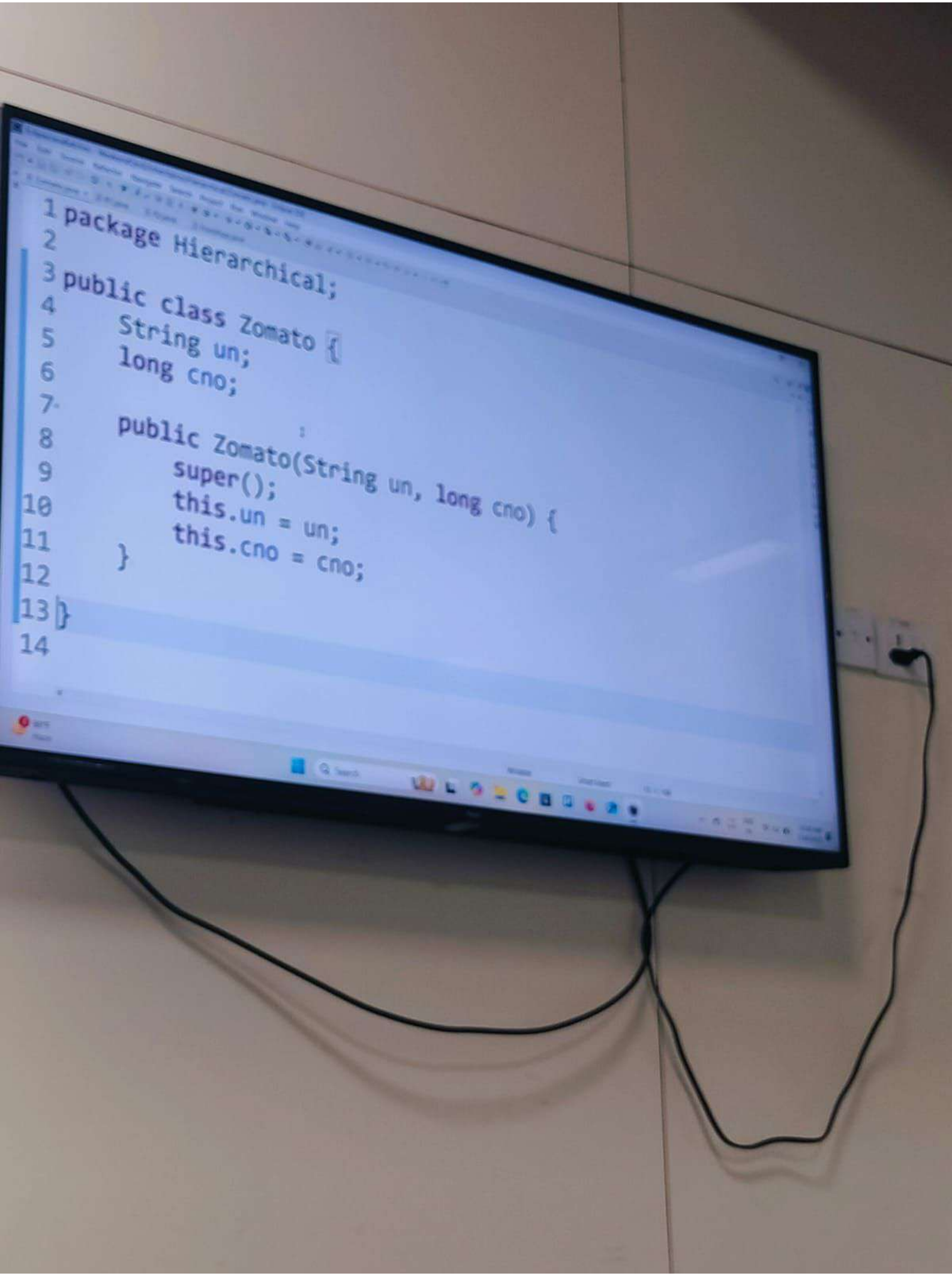

1 package Single;
2
3 public class Bike extends Company {
4
5
6 String bname,model;
7 double price;
8
9 Bike(int wheel, String name,String bname,
10 String model,double price) {
11 super(wheel, name);
12 this.bname=bname;
13 this.model=model;
14 this.price=price;
15 }

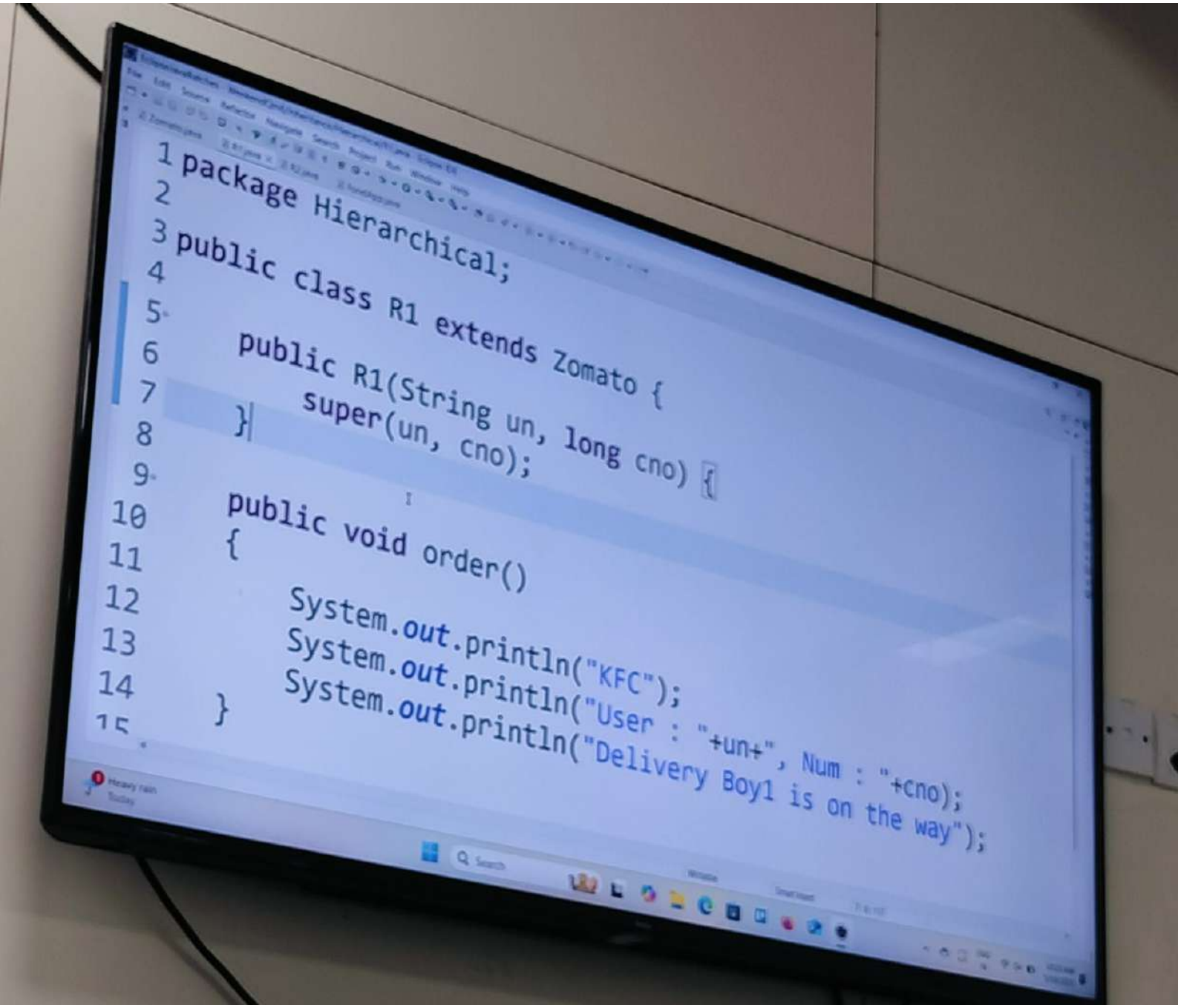


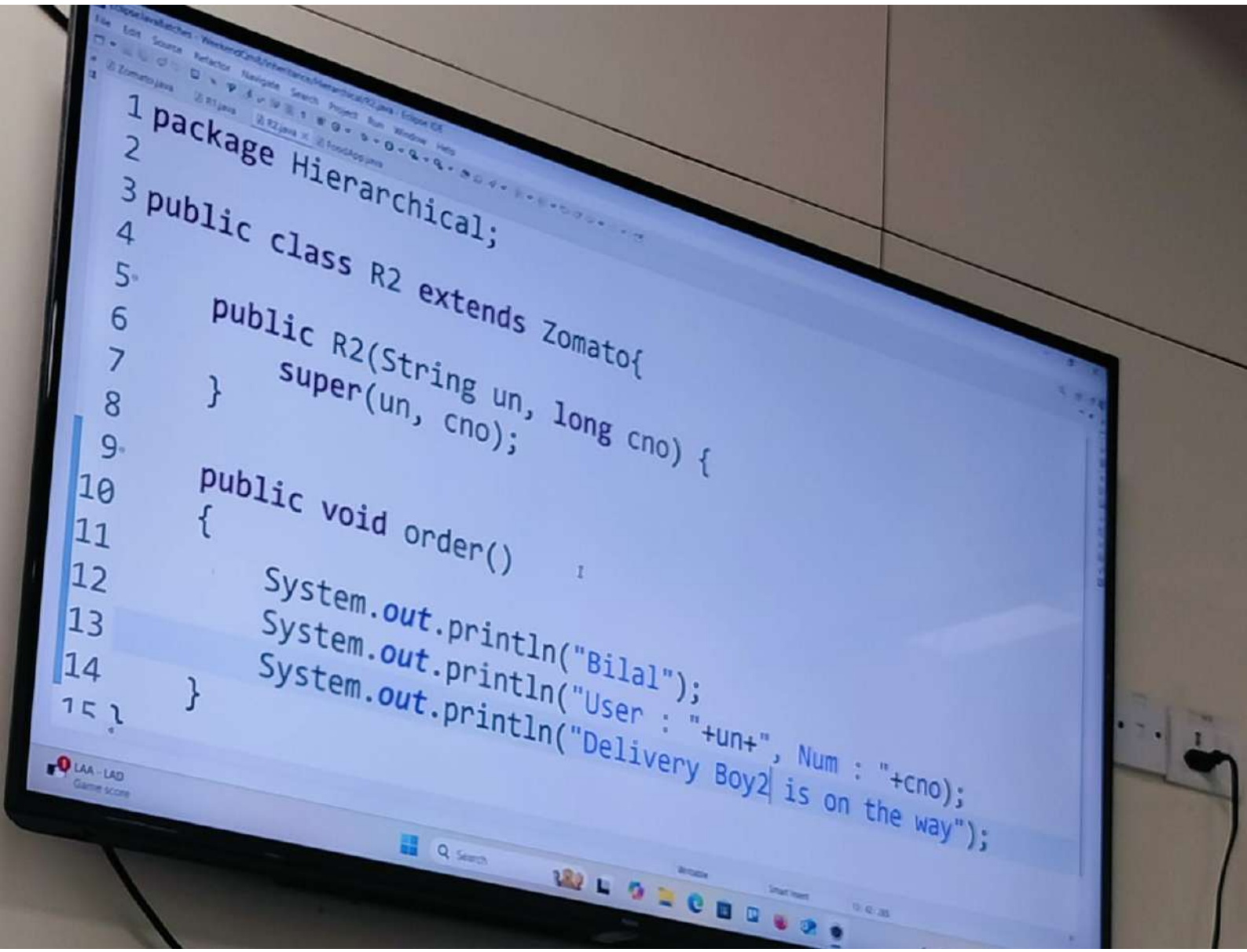
```java
package Single;

public class MainDriver {

    public static void main(String[] args) {
        Bike b1=new Bike(2, "Bajaj", "Pulsar", "NS-200", 120000);
        System.out.println(b1.wheel+" Wheeler");
        System.out.println("Company : "+b1.name);
        System.out.println("Bike Name : "+b1.bname);
        System.out.println("Model : "+b1.model);
        System.out.println("Price : "+b1.price);
    }

}
```

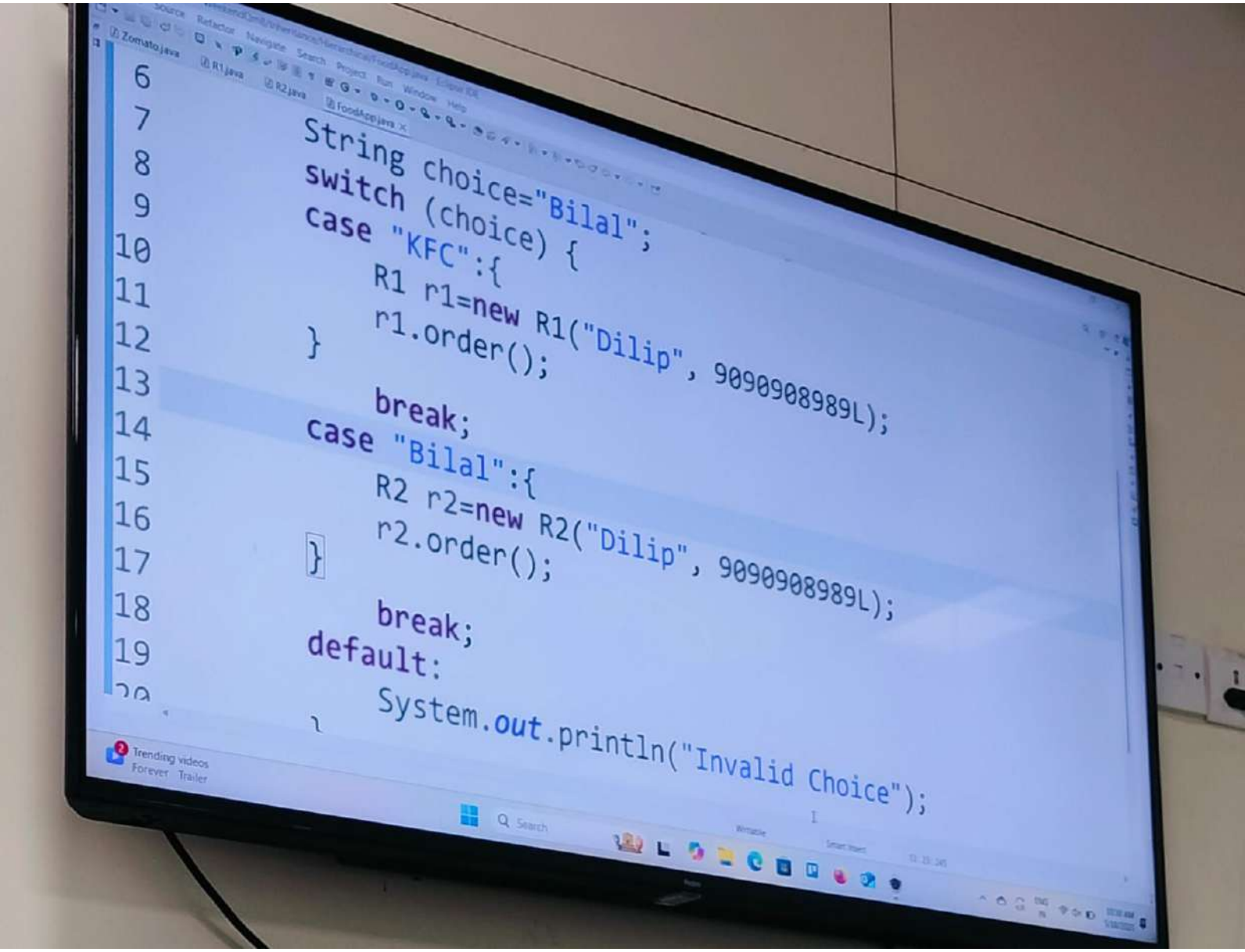

1 package Single;
2
3 public class Company extends Vehicle {
4 String name;
5
6 Company(int wheel,String name)
7 {
8 super(wheel);
9 this.name=name;
10 }
11 }
12

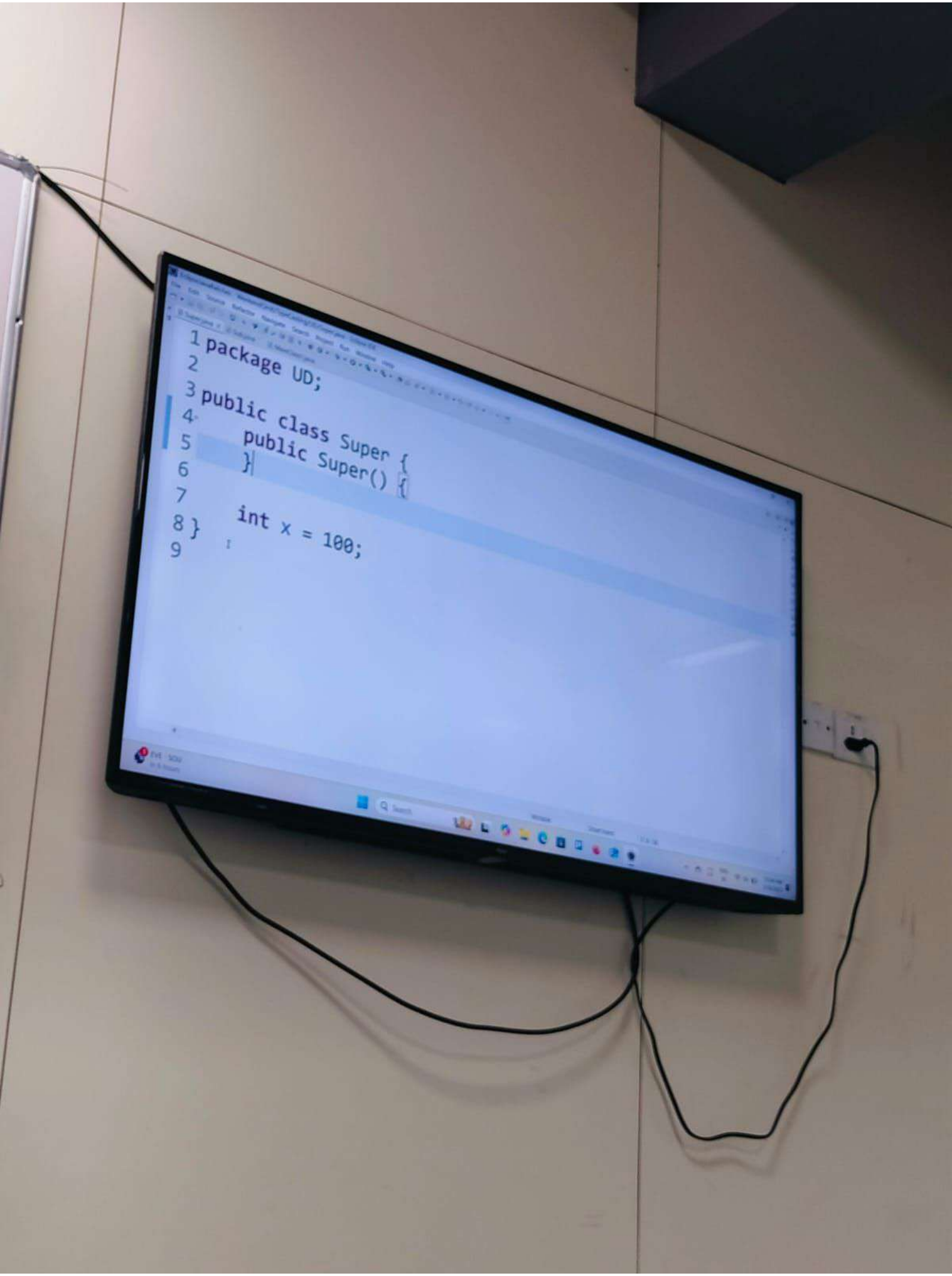

1 package Single;
2
3 public class Vehicle {
4 int wheel;
5 public Vehicle() {
6 System.out.println("Vehicle Types");
7 }
8 public Vehicle(int wheel) {
9 this();
10 //super();//-->Object class
11 this.wheel = wheel;
12 }
13 }
14

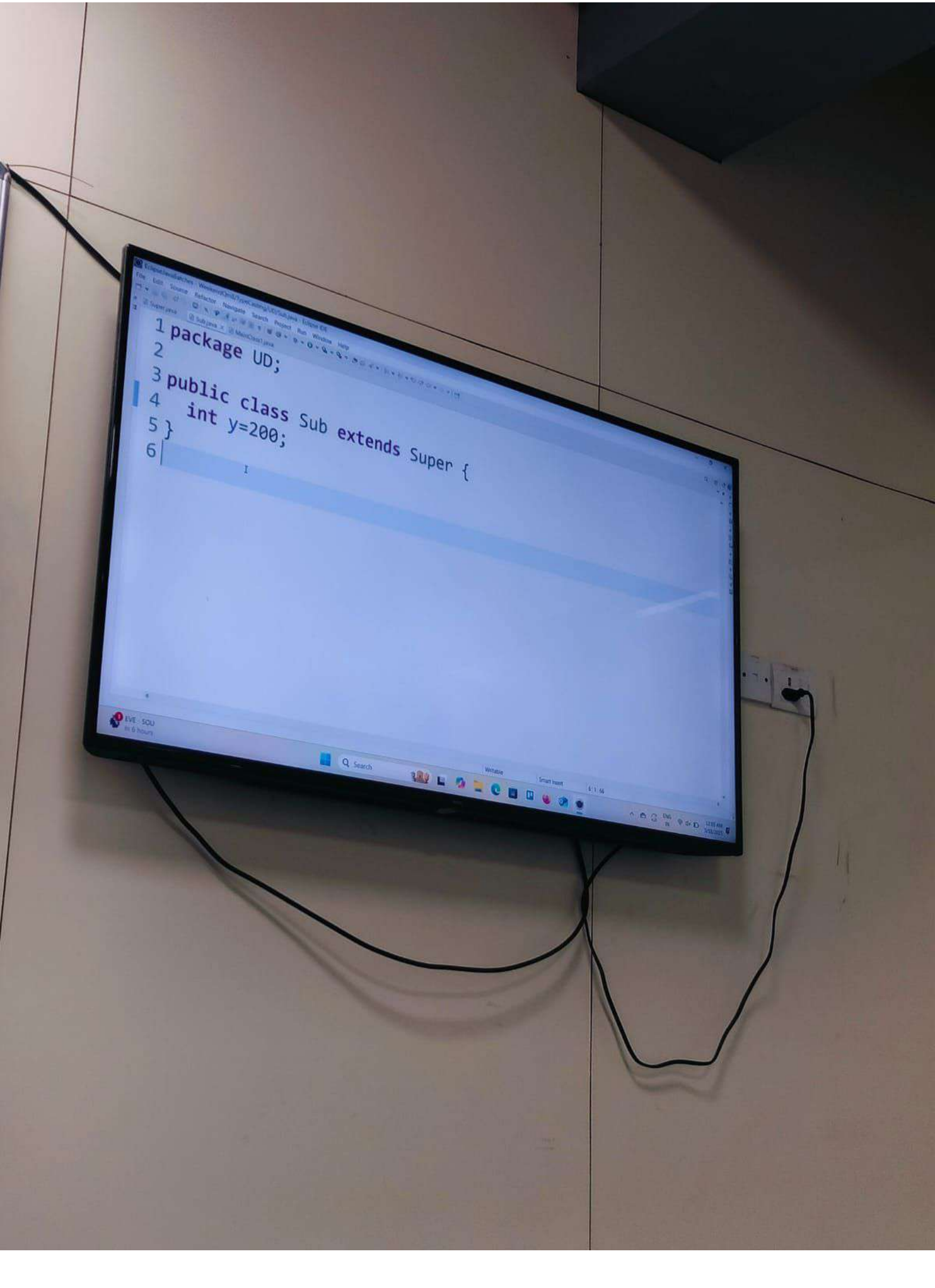

1 package Hierarchical;
2
3 public class Zomato {
4 String un;
5 long cno;
6
7
8 public Zomato(String un, long cno) {
9 super();
10 this.un = un;
11 this.cno = cno;
12 }
13 }
14

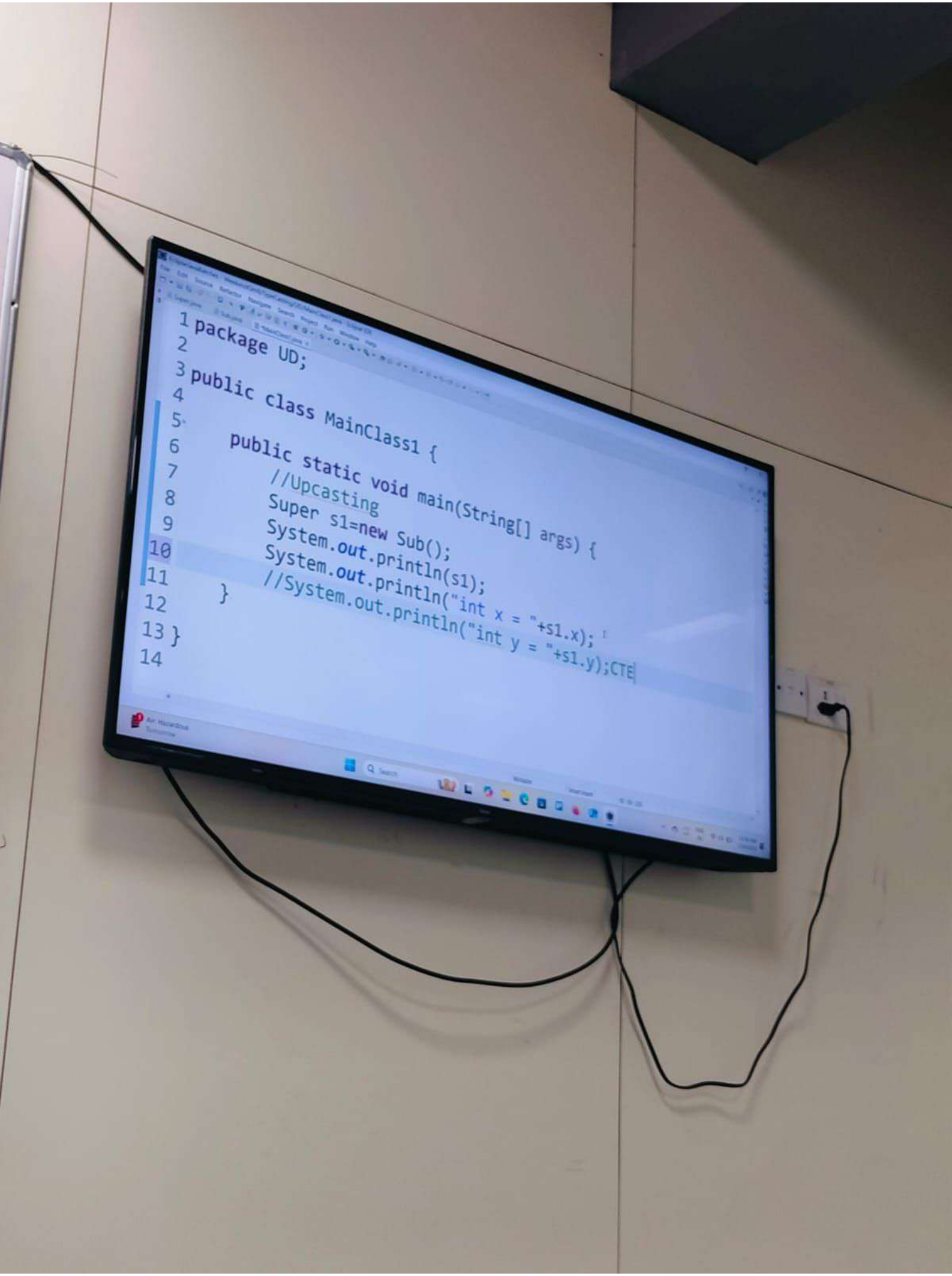

1 package Hierarchical;
2
3 public class R1 extends Zomato {
4
5
6 public R1(String un, long cno) {
7 super(un, cno);
8 }
9
10 public void order()
11 {
12 System.out.println("KFC");
13 System.out.println("User : "+un+", Num : "+cno);
14 System.out.println("Delivery Boy1 is on the way");
15 }

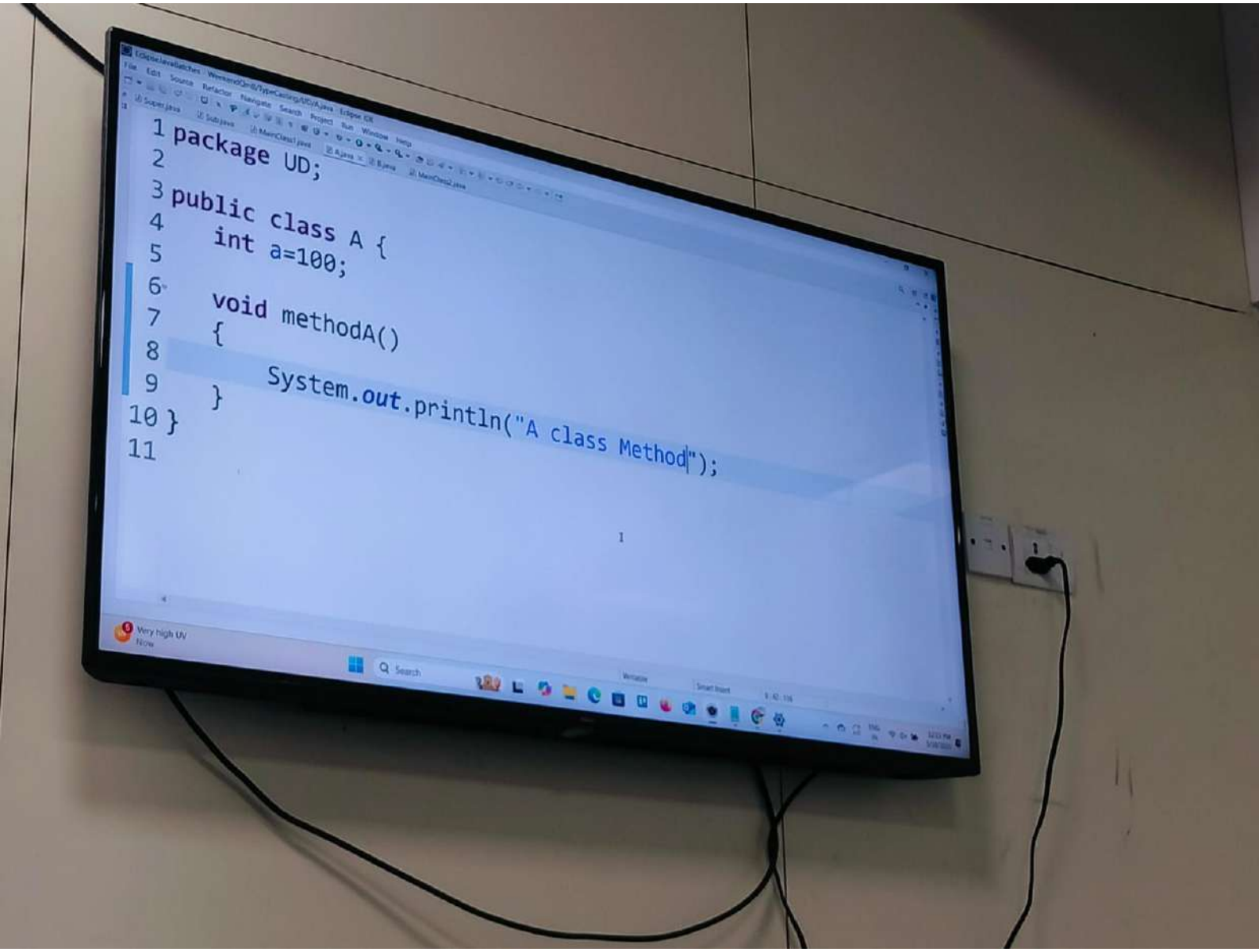

1 package Hierarchical;
2
3 public class R2 extends Zomato{
4
5
6 public R2(String un, long cno) {
7 super(un, cno);
8 }
9
10 public void order()
11 {
12 System.out.println("Bilal");
13 System.out.println("User : "+un+", Num : "+cno);
14 System.out.println("Delivery Boy2 is on the way");
15 }

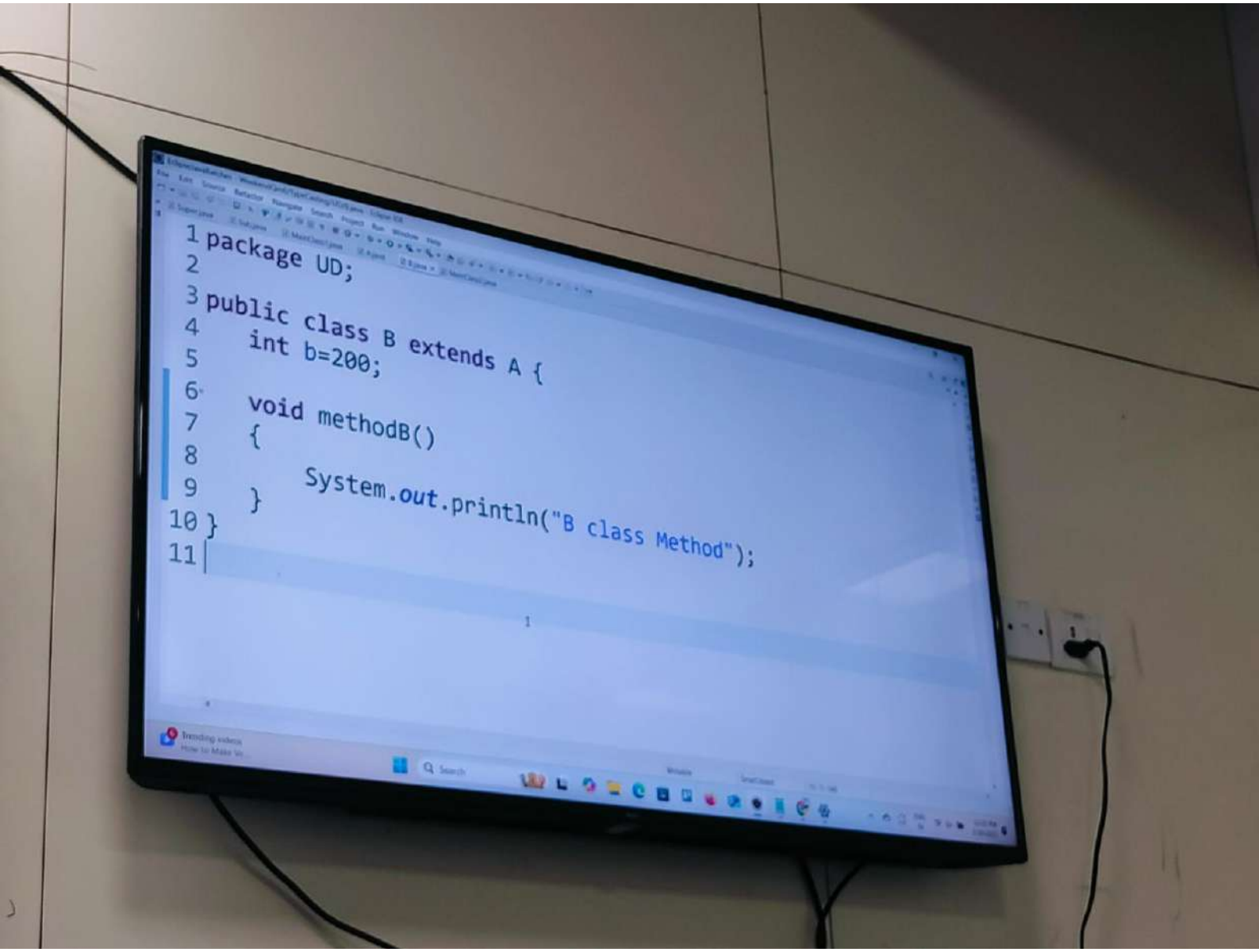

String choice="Bilal";
switch (choice) {
case "KFC":{
R1 r1=new R1("Dilip", 9090908989L);
r1.order();
}
break;
case "Bilal":{
R2 r2=new R2("Dilip", 9090908989L);
r2.order();
}
break;
default:
System.out.println("Invalid Choice");

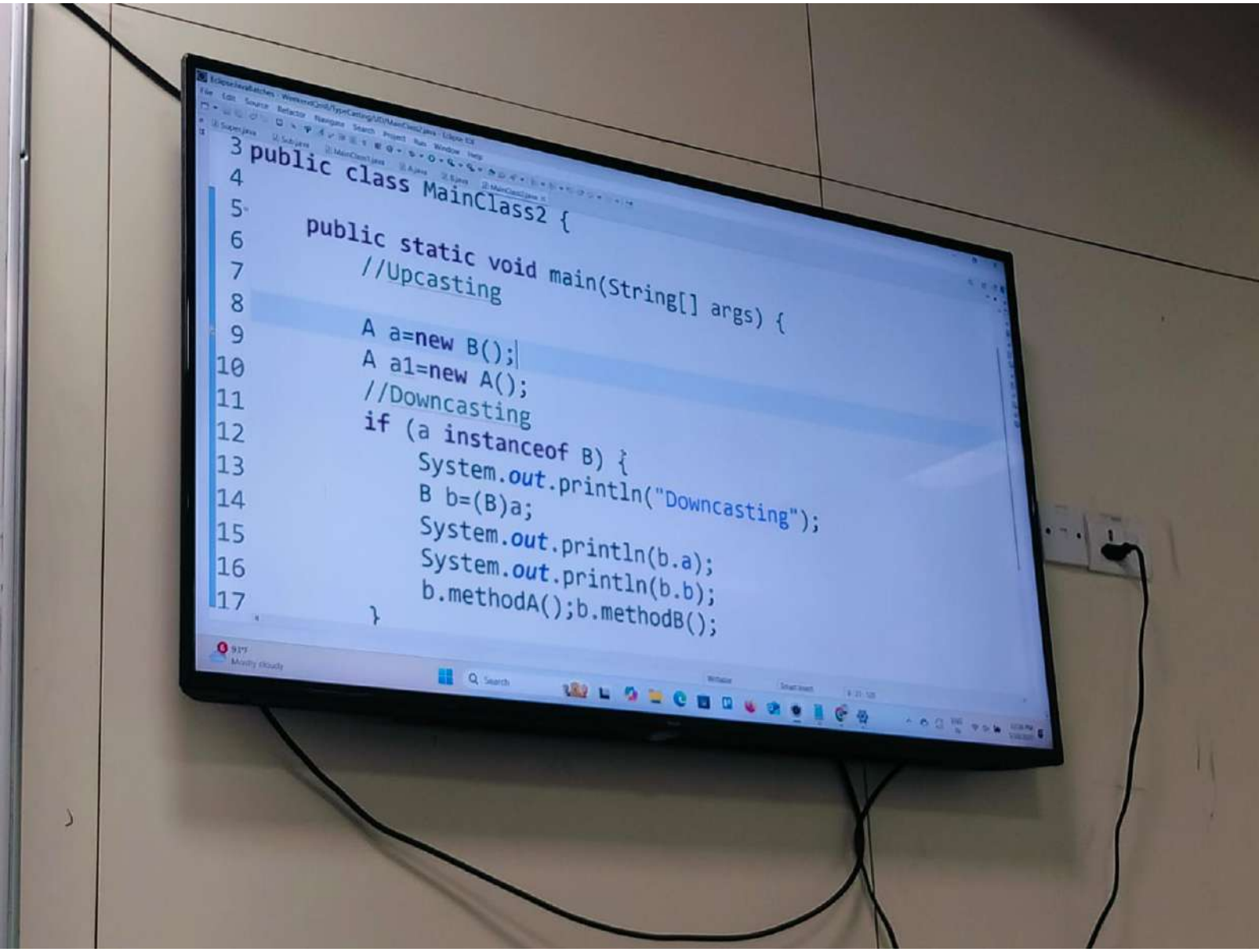

1 package UD;
2
3 public class Super {
4
5 public Super() {
6 }
7
8 }
9 int x = 100;

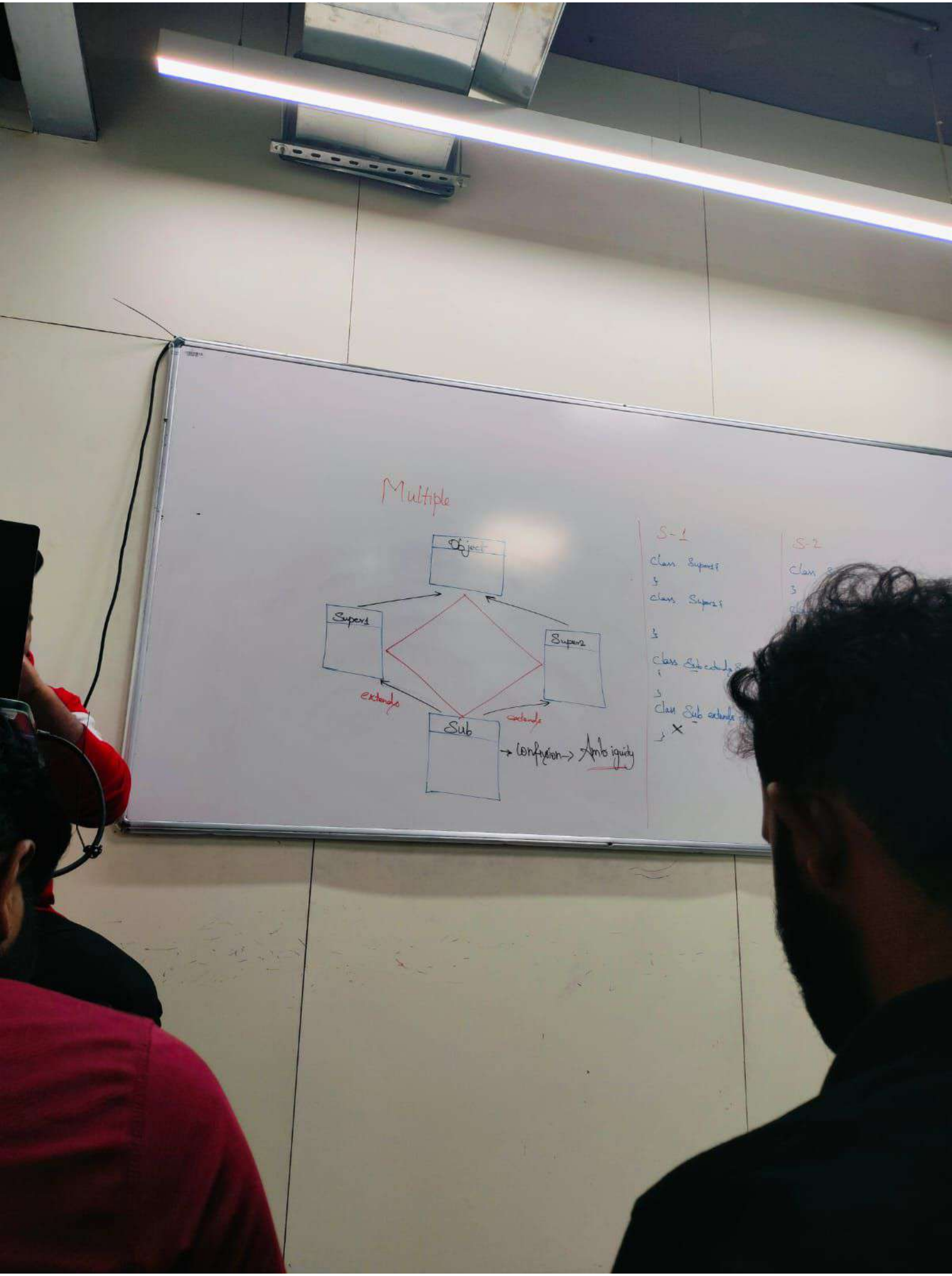

1 package UD;
2
3 public class Sub extends Super {
4 int y=200;
5 }
6


1 package UD;
2
3 public class MainClass1 {
4
5
6 public static void main(String[] args) {
7 //Upcasting
8 Super s1=new Sub();
9 System.out.println(s1);
10 System.out.println("int x = "+s1.x);
11 //System.out.println("int y = "+s1.y);CTE
12 }
13 }
14

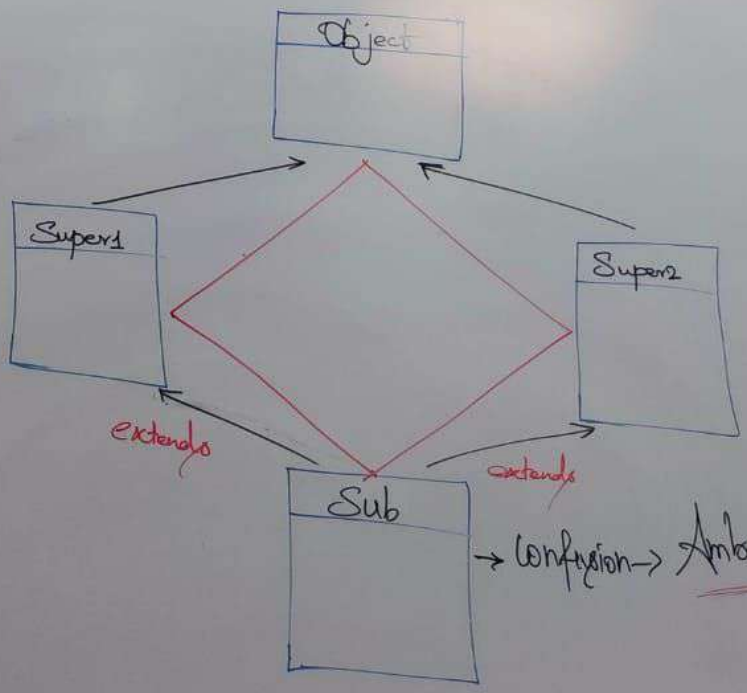

1 package UD;
2
3 public class A {
4 int a=100;
5
6
7 void methodA()
8 {
9 System.out.println("A class Method");
10 }
11 }


1 package UD;
2
3 public class B extends A {
4 int b=200;
5
6
7 void methodB()
8 {
9
System.out.println("B class Method");
10 }
11


public class MainClass2 {
public static void main(String[] args) {
//Upcasting
A a=new B();
A a1=new A();
//Downcasting
if (a instanceof B) {
System.out.println("Downcasting");
B b=(B)a;
System.out.println(b.a);
System.out.println(b.b);
b.methodA();b.methodB();
}


Multiple
Object
Super1
Super2
extends
extends
Sub
→ confusion → Ambiguity
S-1
Class Super1
{
}
class Super2
{
}
Class Sub extends
{
}
Class Sub extends
{
}
X
S-2

Multiple

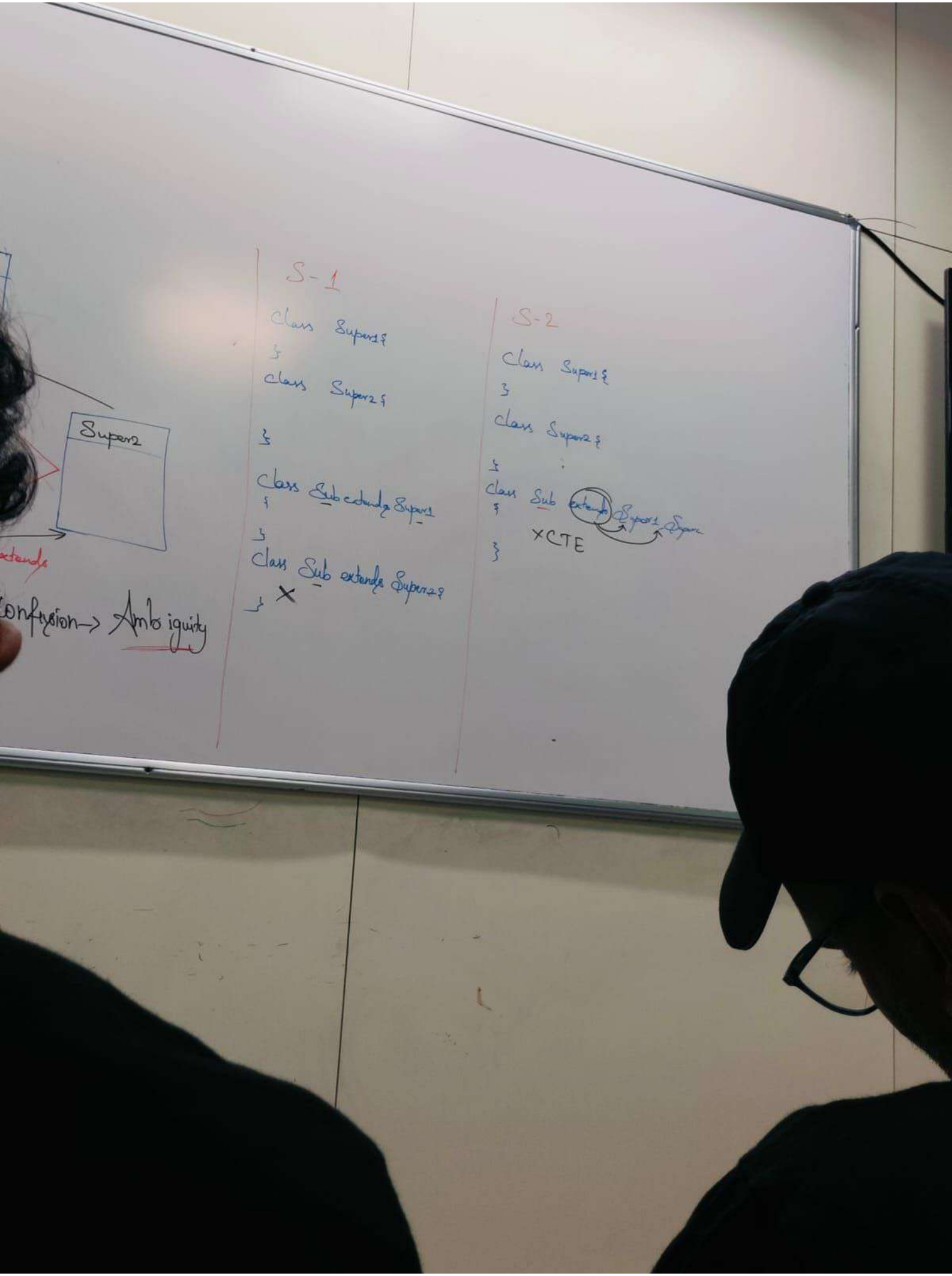


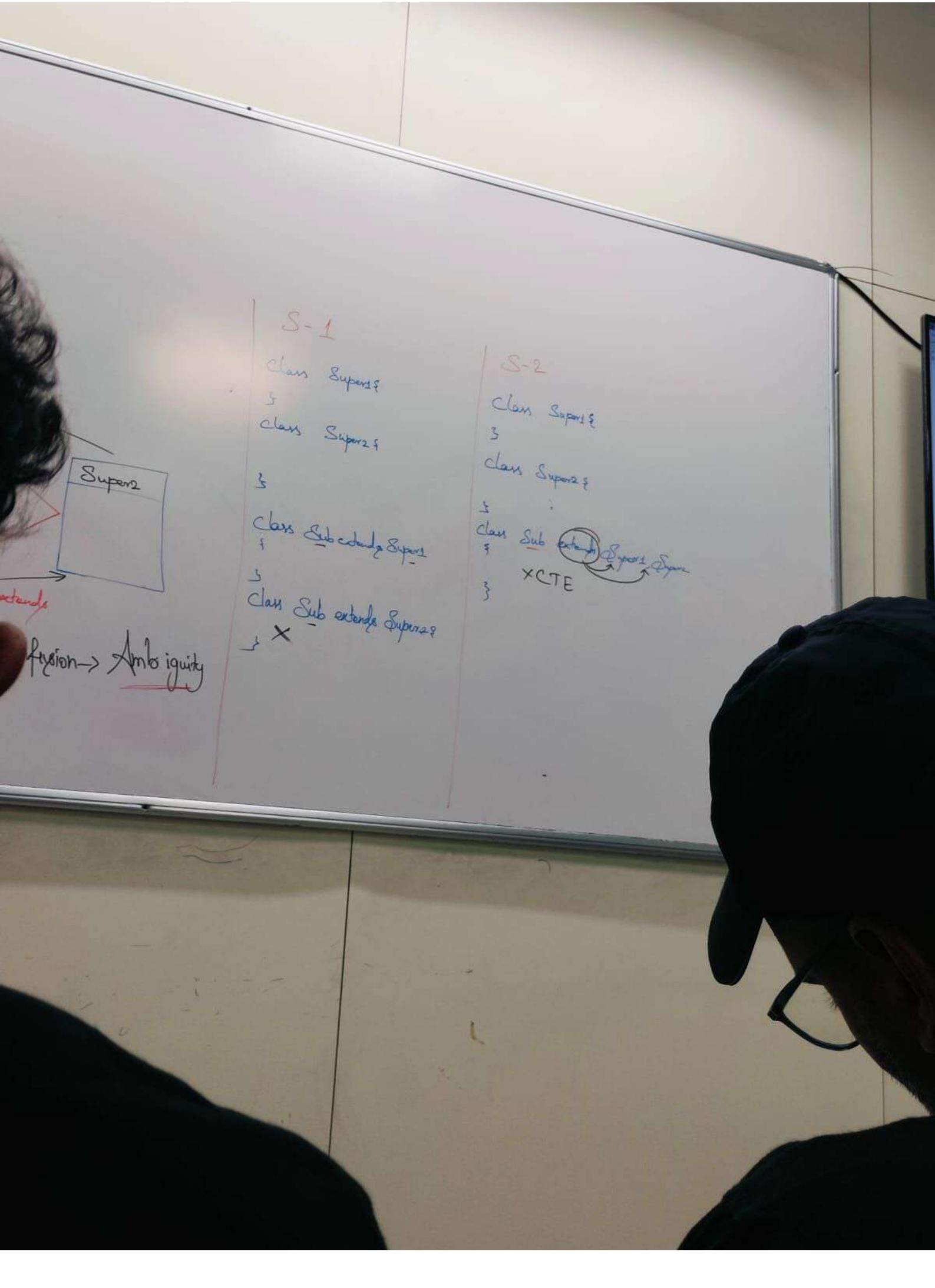
→ Confusion → Ambiguity

S-1

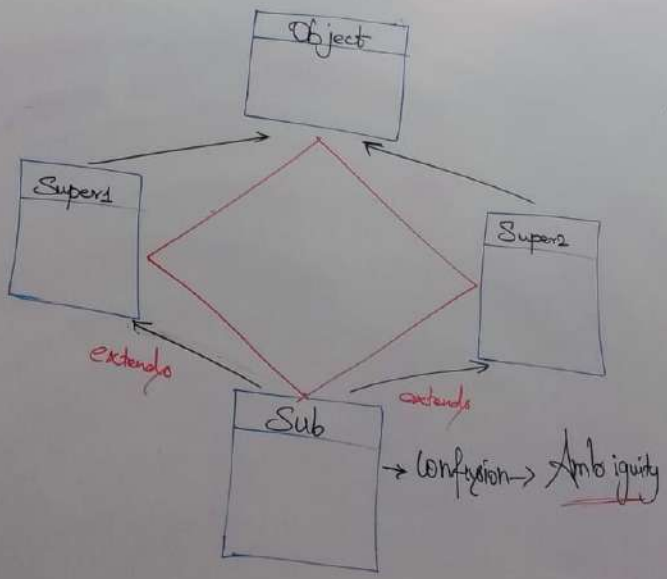
```
class Super1{
}
class Super2{

}
class Sub extend
{
}
class S
}
```

×


Super2
extends
confusion -> Ambiguity
S-1
class Super1 {
}
class Super2 {
}
class Sub extends Super1 {
}
class Sub extends Super2 {
} X
S-2
class Super1 {
}
class Super2 {
}
class Sub extends Super1, Super2
{
}
XCTE


Super2
extends
fusion -> Ambiguity
S-1
class Super1 {
}
class Super2 {
}
class Sub extends Super1
{
}
class Sub extends Super2
{
}
X
S-2
class Super1 {
}
class Super2 {
}
class Sub extends Super1 Super2
{
}
XCTE

Multiple



S-1

```
class Super1 {
}
class Super2 {

}
class Sub extends Super1
{

}
class Sub extends Super2 {
}
```

X

S-2

```
class Super1 {
}
class Super2 {

}
class Sub extends Super1, Super2
{
}
```

XCTE

Non Primitive typecasting

→ Convert one class into another
Class

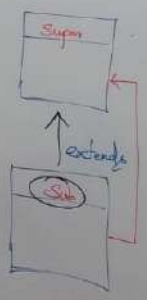
2 types

✓* Upcasting

✓* Downcasting

→ It should be inherit first

Upcasting

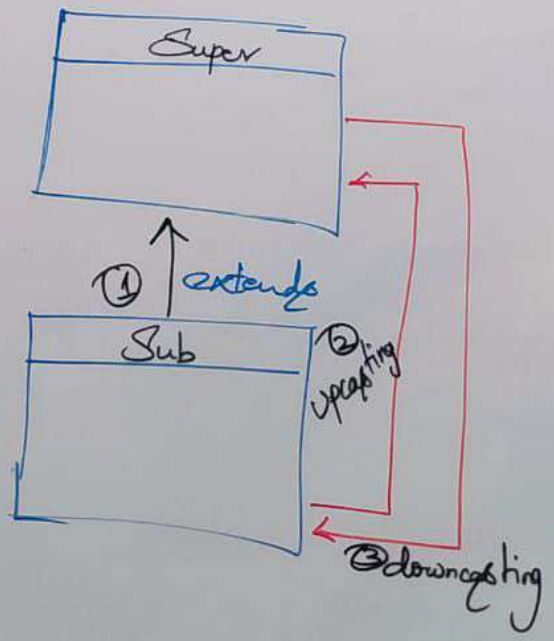


Class Super {
int x = 100;
}

Class Sub extends Super {
→ 1 var is inherited
int y = 200;
}
Sub class properties

Class Main {
P.S.V.m(SC[]args)
{
// Upcasting
Super A = new Sub();
✓ Sopln(A.x); // 100
× Sopln(A.y); // CTE
}
}

Downcasting



✗ ① Sub s1 = new Super();
Super@ 0x1f3

↳ CTE

② Sub s1 = (Sub) new Super();

↳ CTS

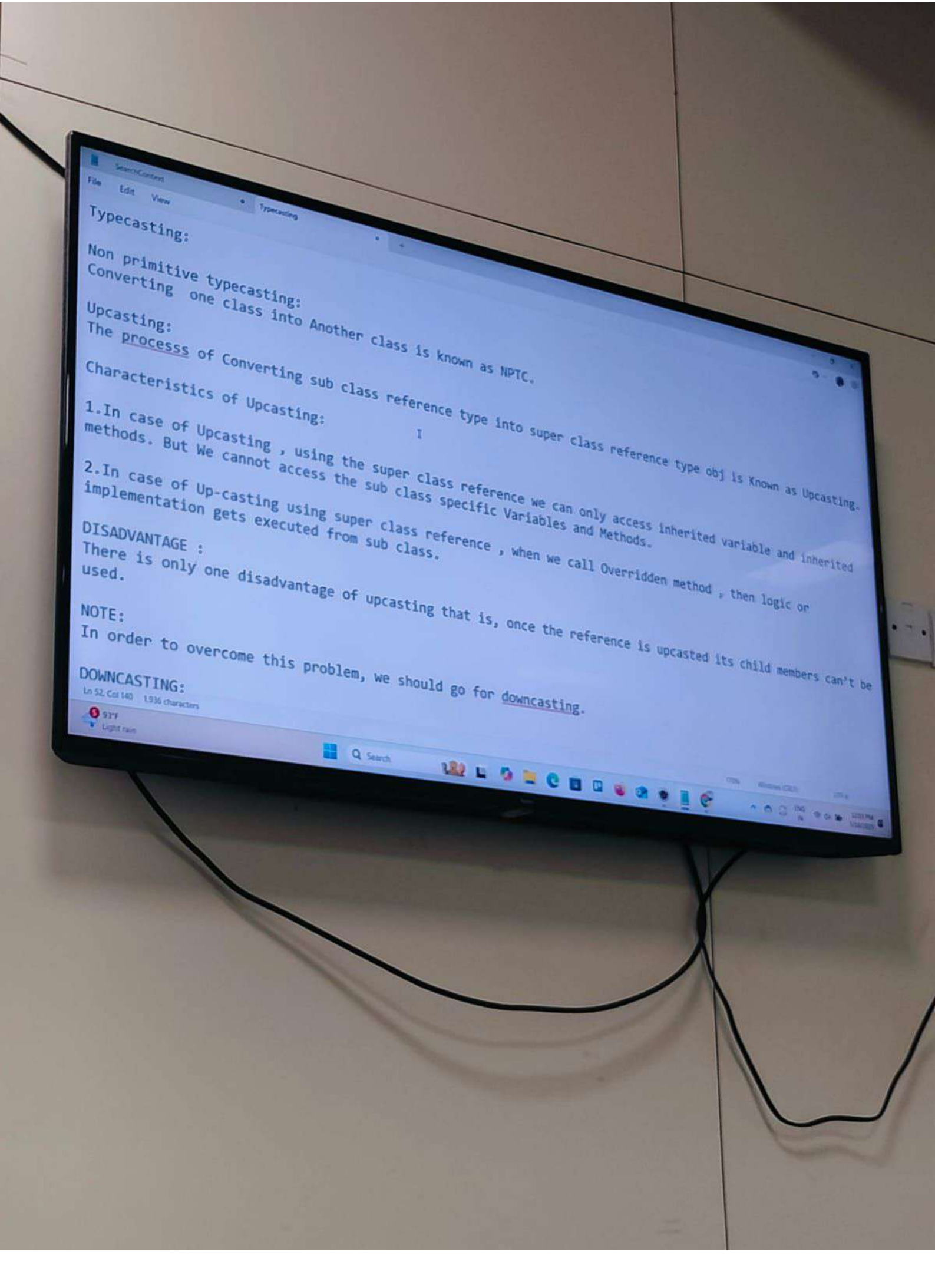
↳ RTException

(ClassCastException)

③ Sub s1 = (Sub) Super Class. Obj;

↳ Subclass Add

Upcasting


File Edit View
Typecasting
Typecasting:
Non primitive typecasting:
Converting one class into Another class is known as NPTC.
Upcasting:
The processs of Converting sub class reference type into super class reference type obj is Known as Upcasting.
Characteristics of Upcasting:
1.In case of Upcasting , using the super class reference we can only access inherited variable and inherited methods. But We cannot access the sub class specific Variables and Methods.
2.In case of Up-casting using super class reference , when we call Overridden method , then logic or implementation gets executed from sub class.
DISADVANTAGE :
There is only one disadvantage of upcasting that is, once the reference is upcasted its child members can't be used.
NOTE:
In order to overcome this problem, we should go for downcasting.
DOWNCASTING:
Ln 52, Col 140 1,936 characters
93°F
Light rain
Search

DOWNCASTING:
The process of converting a parent (superclass) reference type to a child (subclass) reference type is known as downcasting.

NOTE:
Downcasting is not implicitly done by the compiler.
It should be done explicitly by the programmer with the help of a typecast operator.

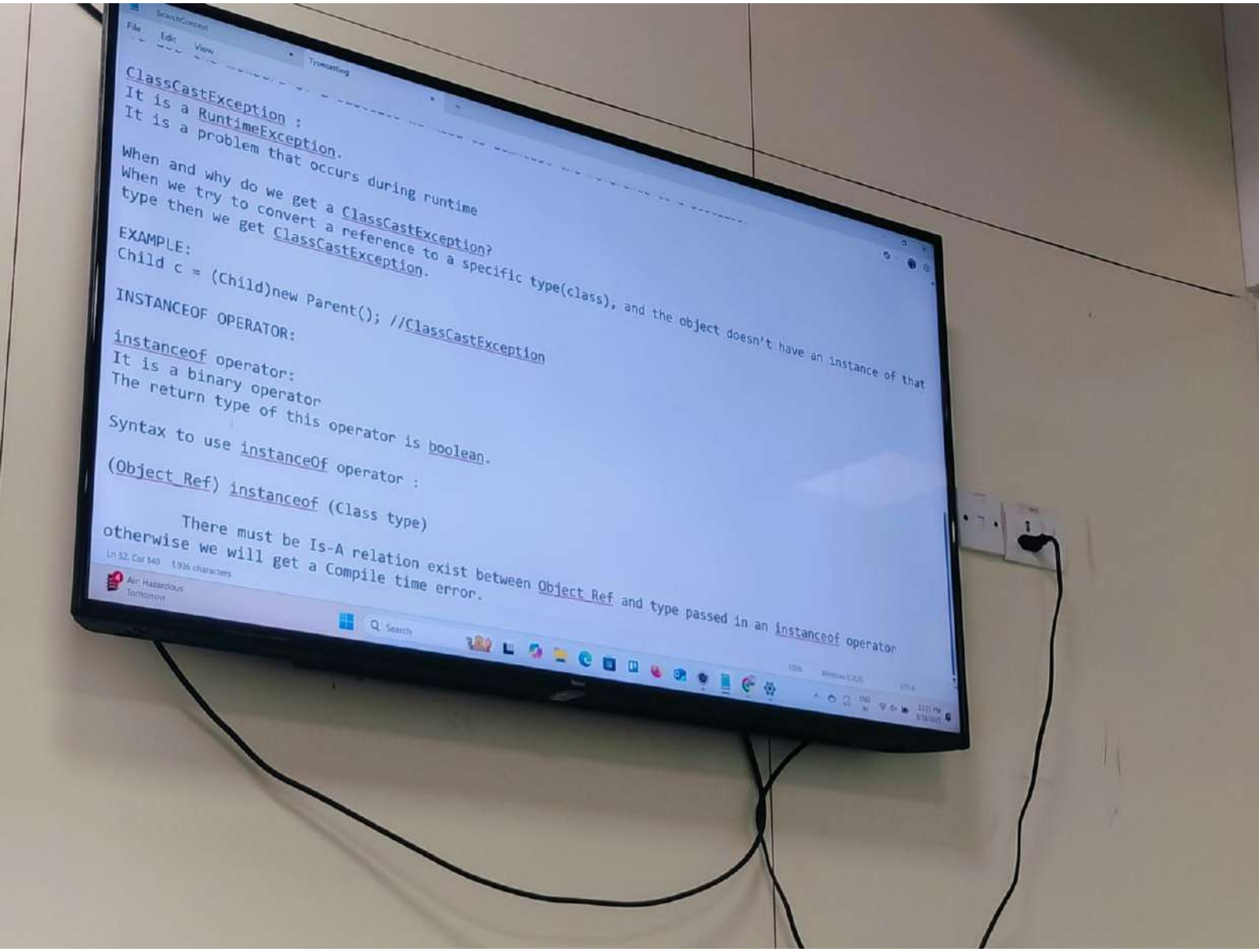
WHY DO WE NEED A DOWNCASTING?
If the reference is upcasted, we can't use the members of a subclass.
To use the members of a subclass we need to downcast the reference to a subclass.

ClassCastException :
It is a RuntimeException.
It is a problem that occurs during runtime

When and why do we get a ClassCastException?
When we try to convert a reference to a specific type(class), and the object doesn't have an instance of that type then we get ClassCastException.

EXAMPLE:
Child c = (Child)new Parent(); //ClassCastException

INSTANCEOF OPERATOR:


ClassCastException :
It is a RuntimeException.
It is a problem that occurs during runtime
When and why do we get a ClassCastException?
When we try to convert a reference to a specific type(class), and the object doesn't have an instance of that type then we get ClassCastException.
EXAMPLE:
Child c = (Child)new Parent(); //ClassCastException
INSTANCEOF OPERATOR:
instanceof operator:
It is a binary operator
The return type of this operator is boolean.
Syntax to use instanceOf operator :
(Object_Ref) instanceof (Class type)
There must be Is-A relation exist between Object_Ref and type passed in an instanceof operator otherwise we will get a Compile time error.