Object Oriented Programming Concepts I

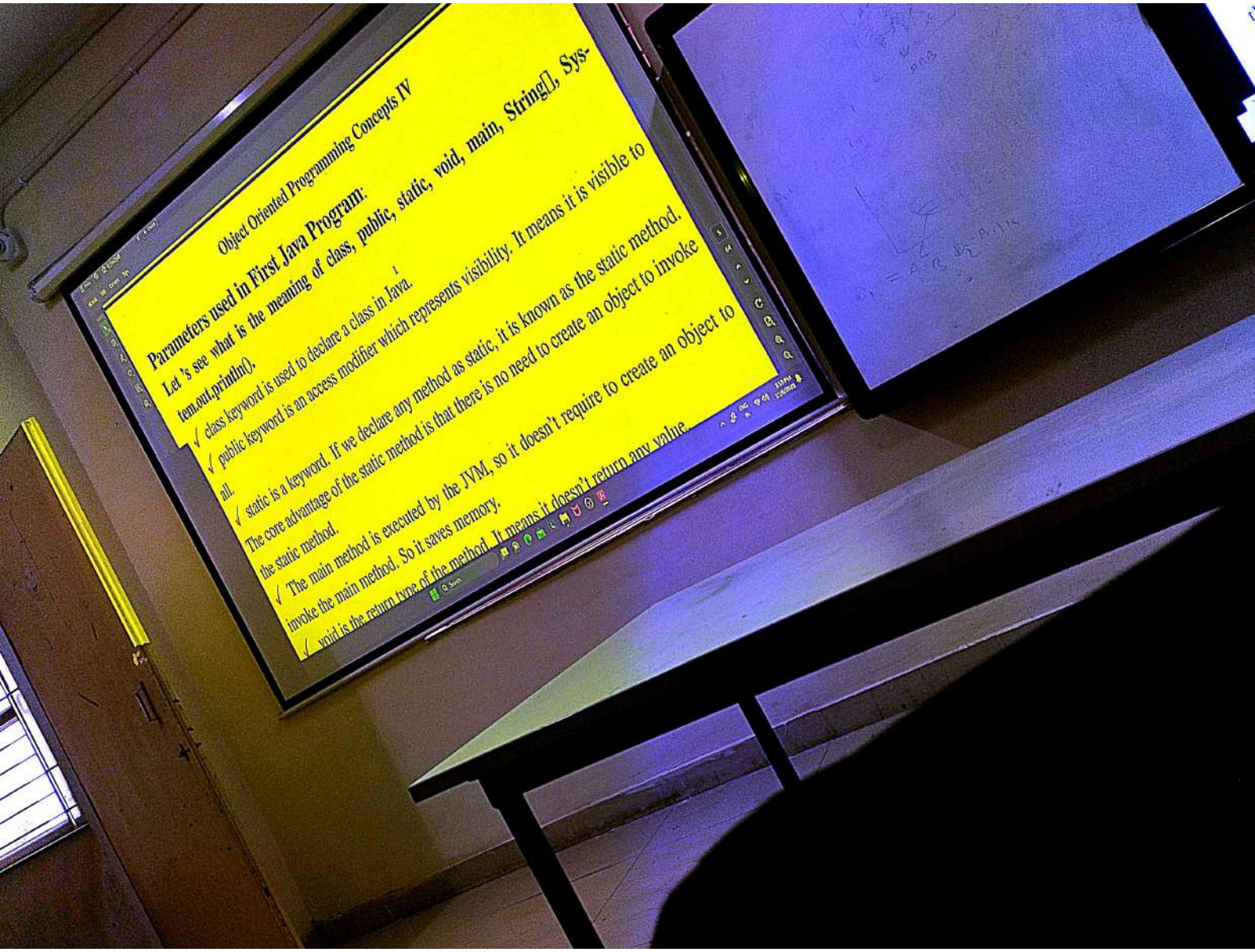
First Simple program in Java

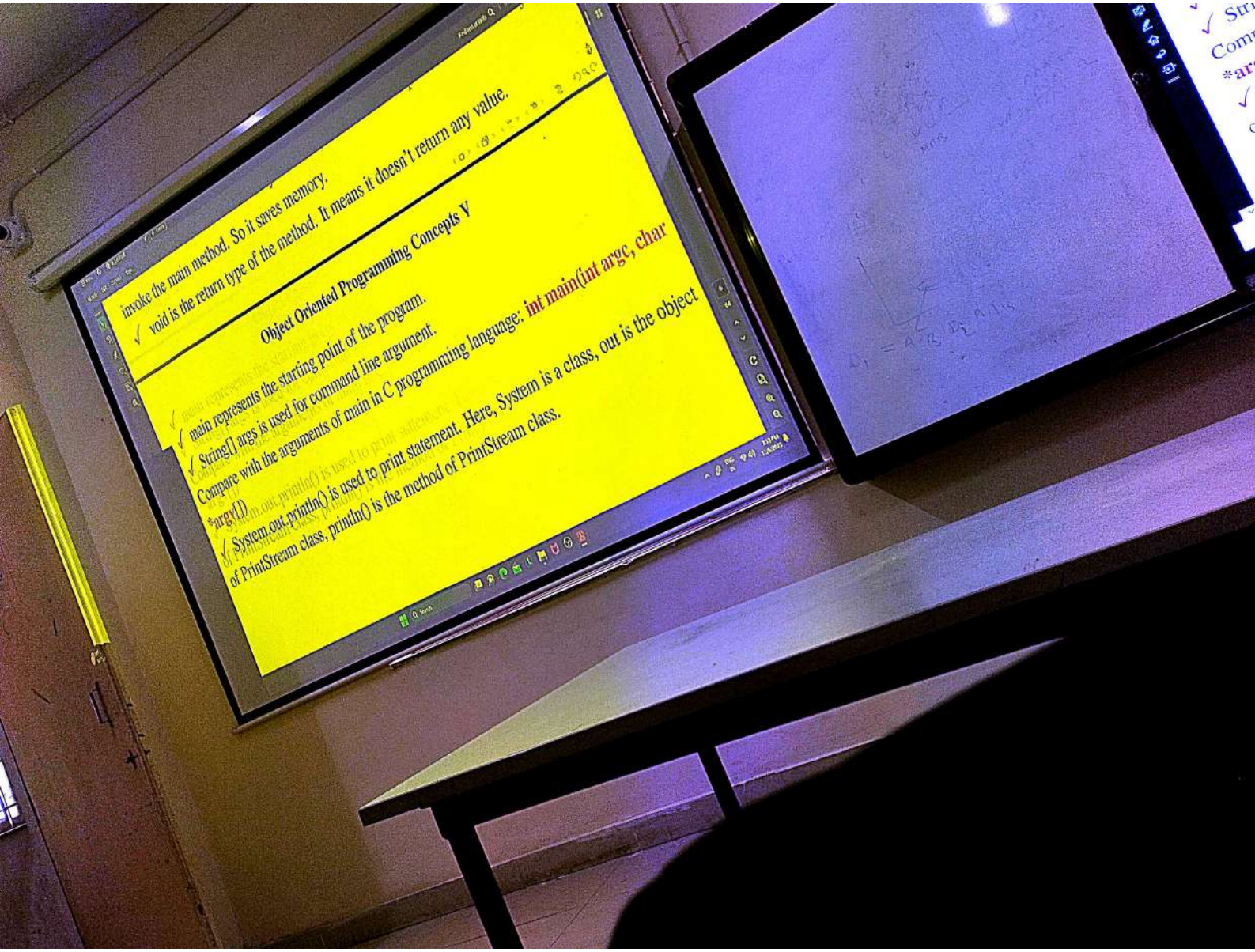
```
public class Simple
{
    public static void main (String args [])
    {
        System.out.println("Hello Java");
    }
}
```

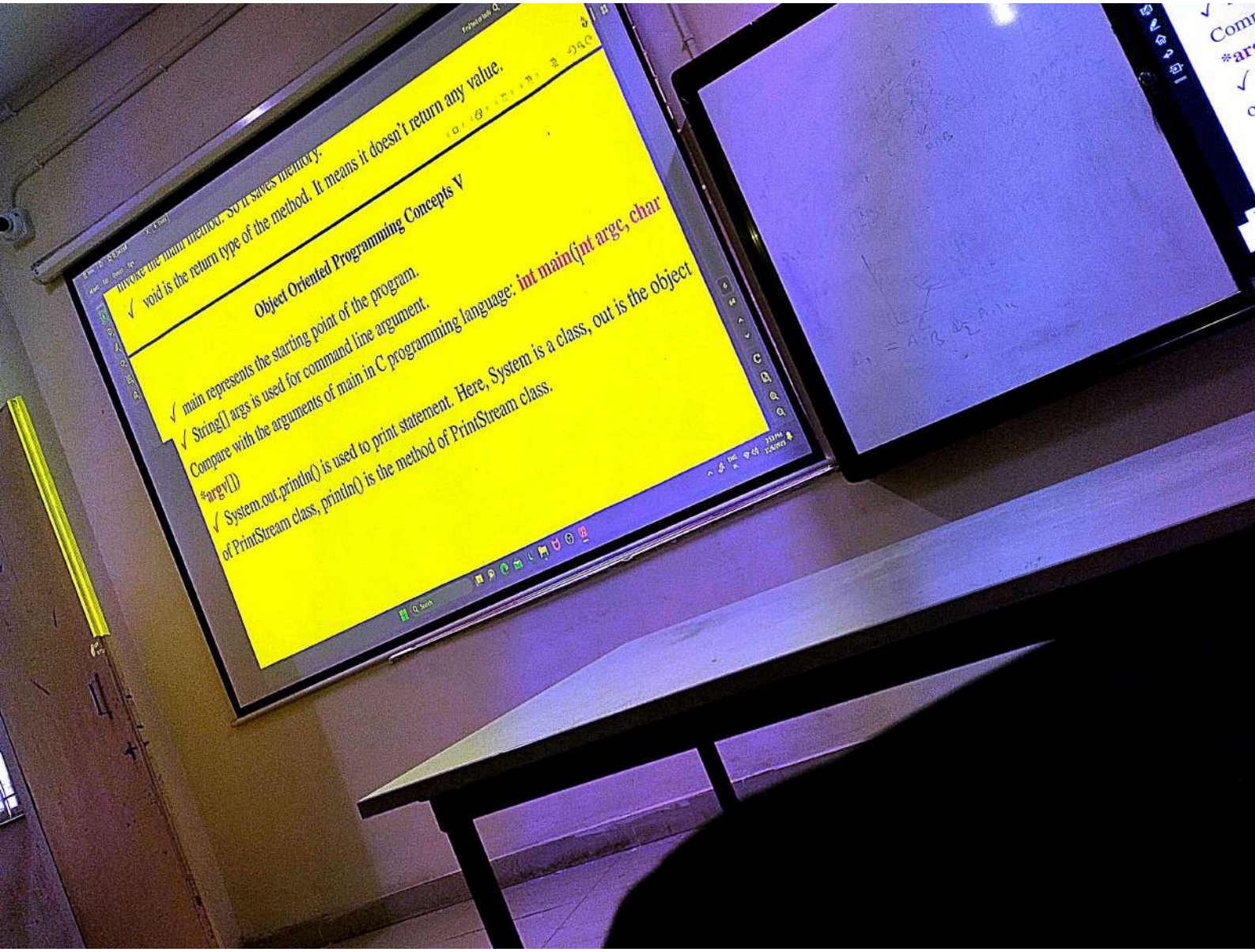
To setup Javac path: https://www.javatpoint.com/how-to-set-path-in-java


Object Oriented Programming Concepts II
Makes it class method so that it can be called using class name without creating an object of the class.
public
static
void
main(String[ ] args)
To call by JVM from anywhere
main method does not return value to JVM.
Name of the method which is called by JVM.
The main() method accepts one argument of type String array.
Fig: Java main method

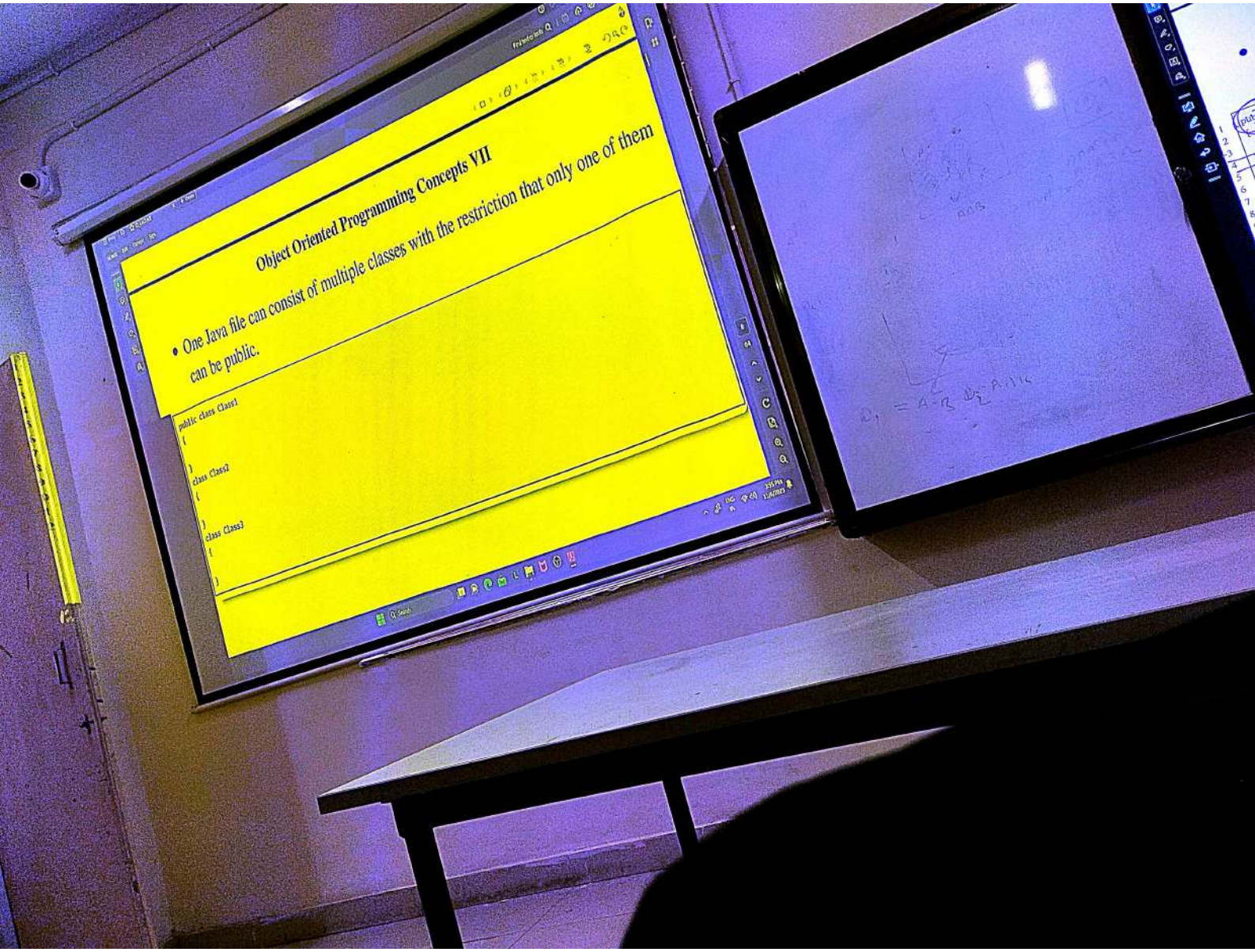

Public Static void main
Execution of program starts from here
Does not return anything
No need to create its object
Accessible outside class
The order in which they occur is not important

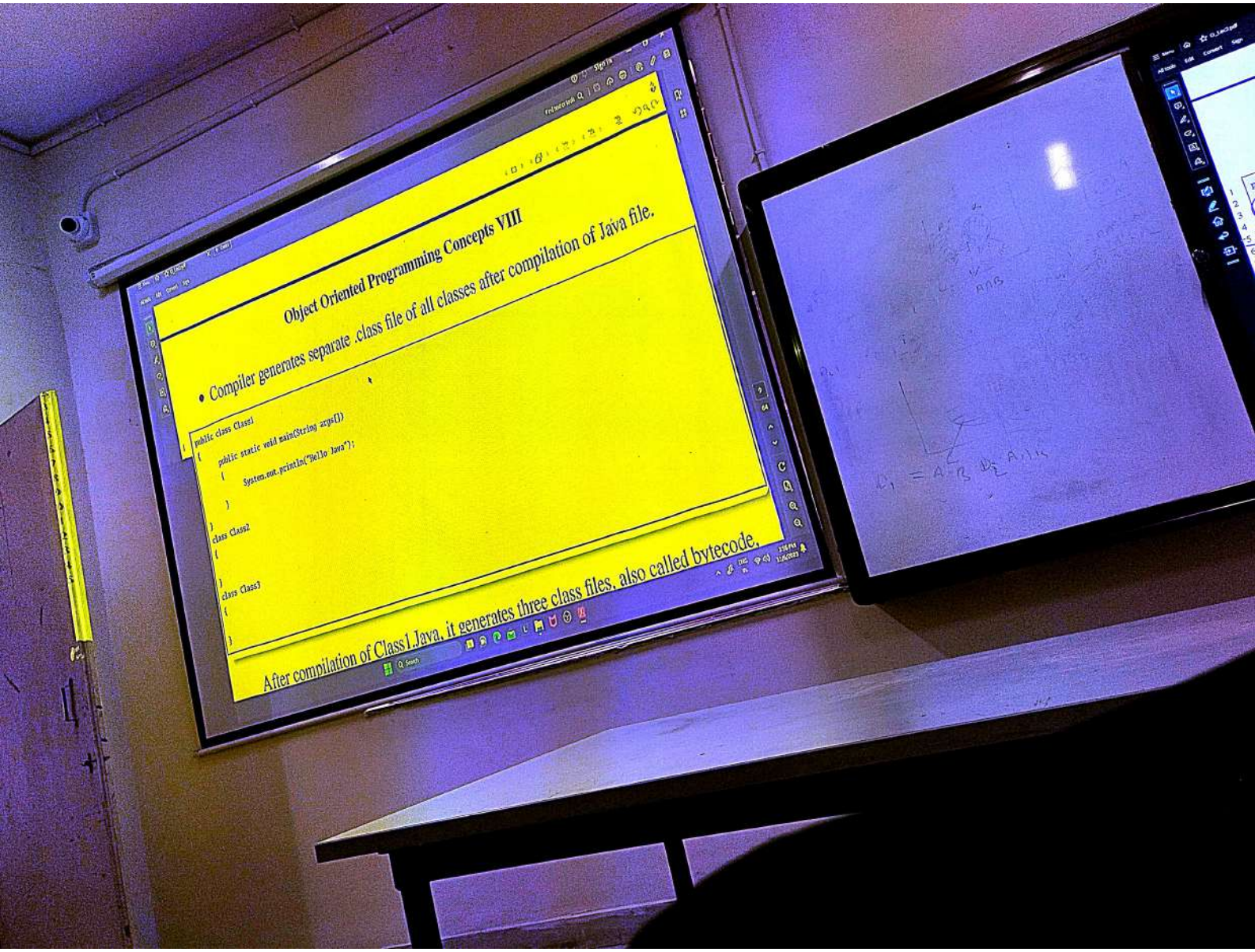

Object Oriented Programming Concepts IV
Parameters used in First Java Program:
Let 's see what is the meaning of class, public, static, void, main, String[], Sys-
tem.out.println().
√ class keyword is used to declare a class in Java.
√ public keyword is an access modifier which represents visibility. It means it is visible to
all.
√ static is a keyword. If we declare any method as static, it is known as the static method.
The core advantage of the static method is that there is no need to create an object to invoke
the static method.
√ The main method is executed by the JVM, so it doesn't require to create an object to
invoke the main method. So it saves memory.
√ void is the return type of the method. It means it doesn't return any value.

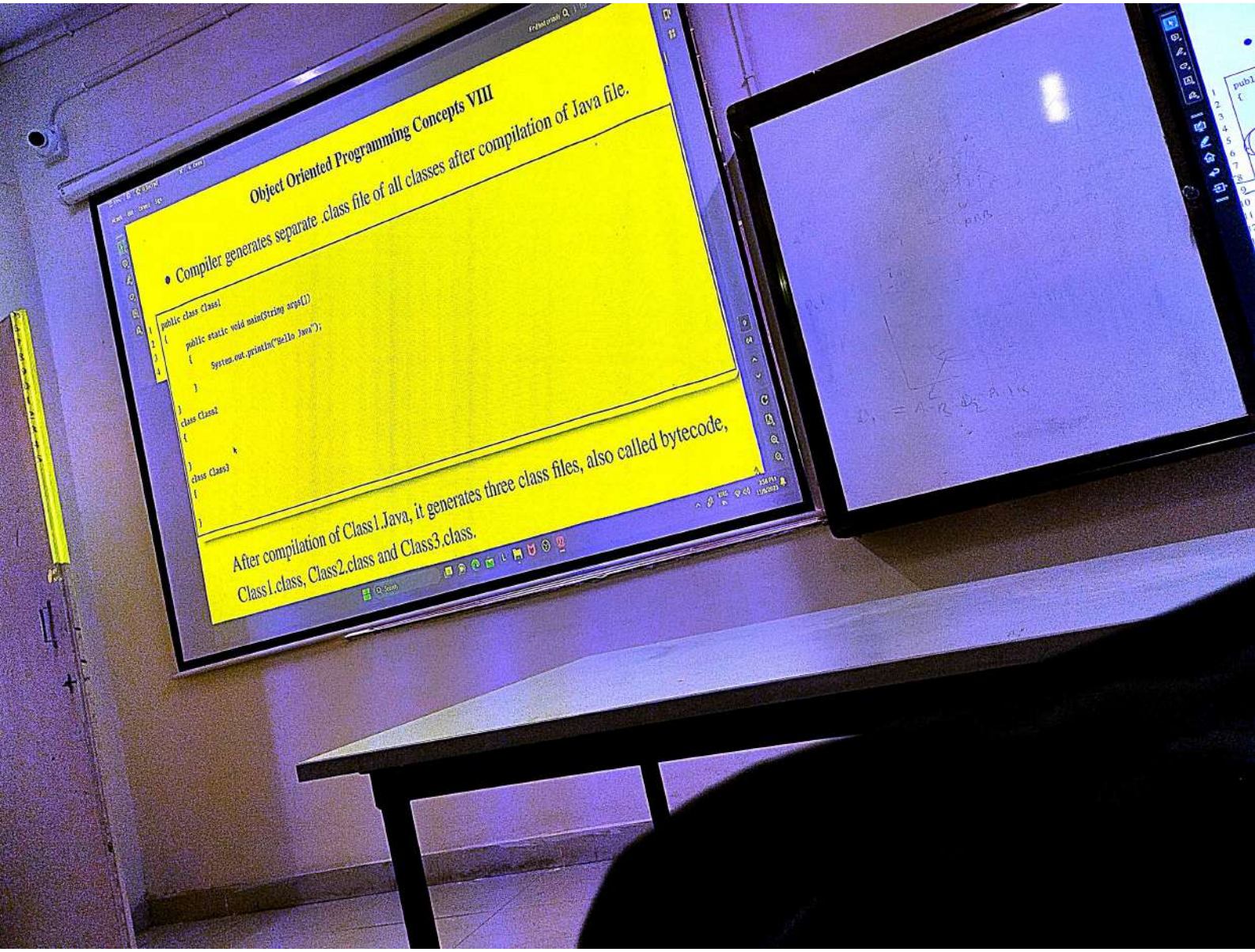

invoke the main method. So it saves memory.
√ void is the return type of the method. It means it doesn't return any value.
Object Oriented Programming Concepts V
√ main represents the starting point of the program.
√ String[] args is used for command line argument.
Compare with the arguments of main in C programming language: int main(int argc, char *argv[])
√ System.out.println() is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

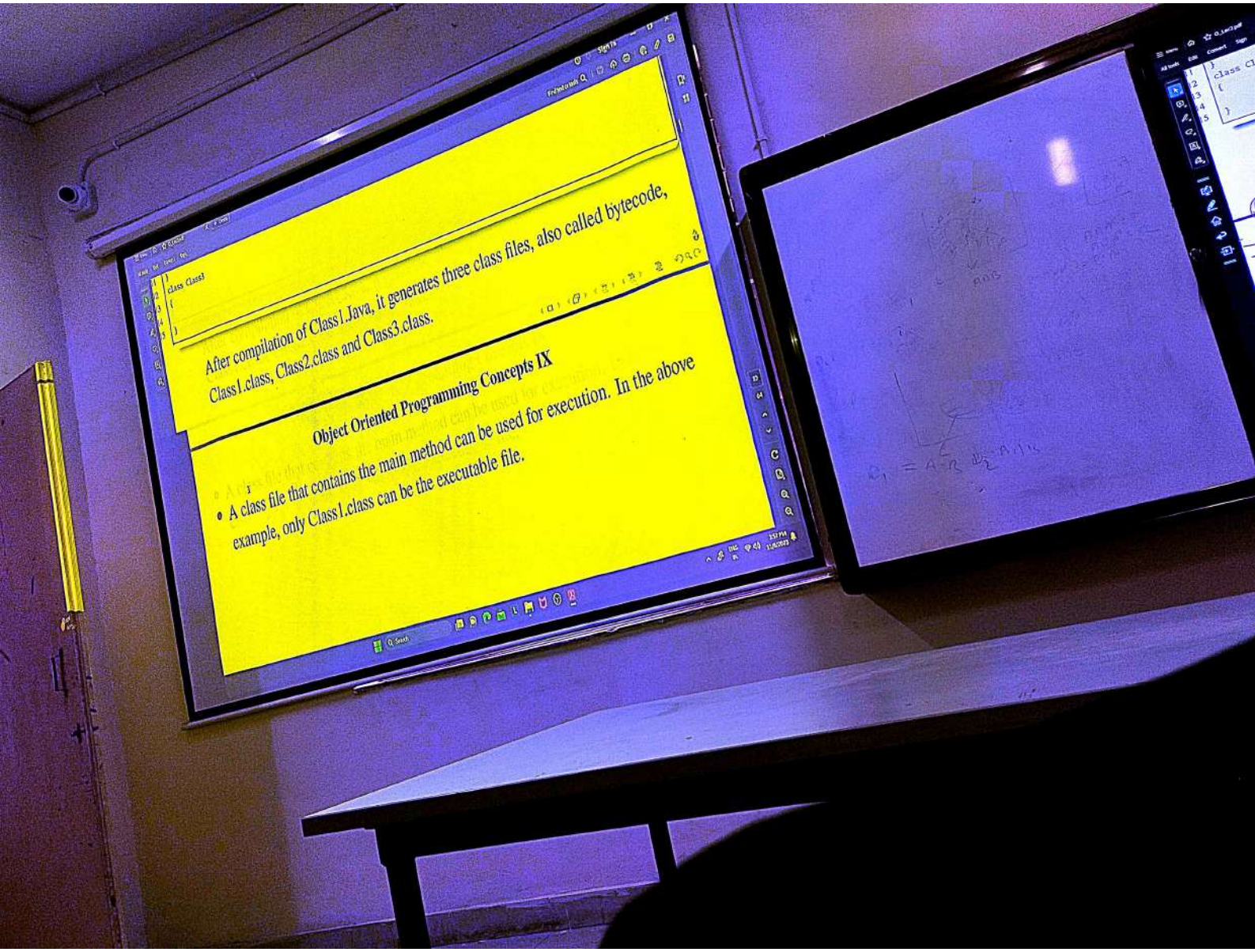

Invoke the main method. So it saves memory.
✓ void is the return type of the method. It means it doesn't return any value.
Object Oriented Programming Concepts V
✓ main represents the starting point of the program.
✓ String[] args is used for command line argument.
Compare with the arguments of main in C programming language: int main(int argc, char *argv[])
✓ System.out.println() is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

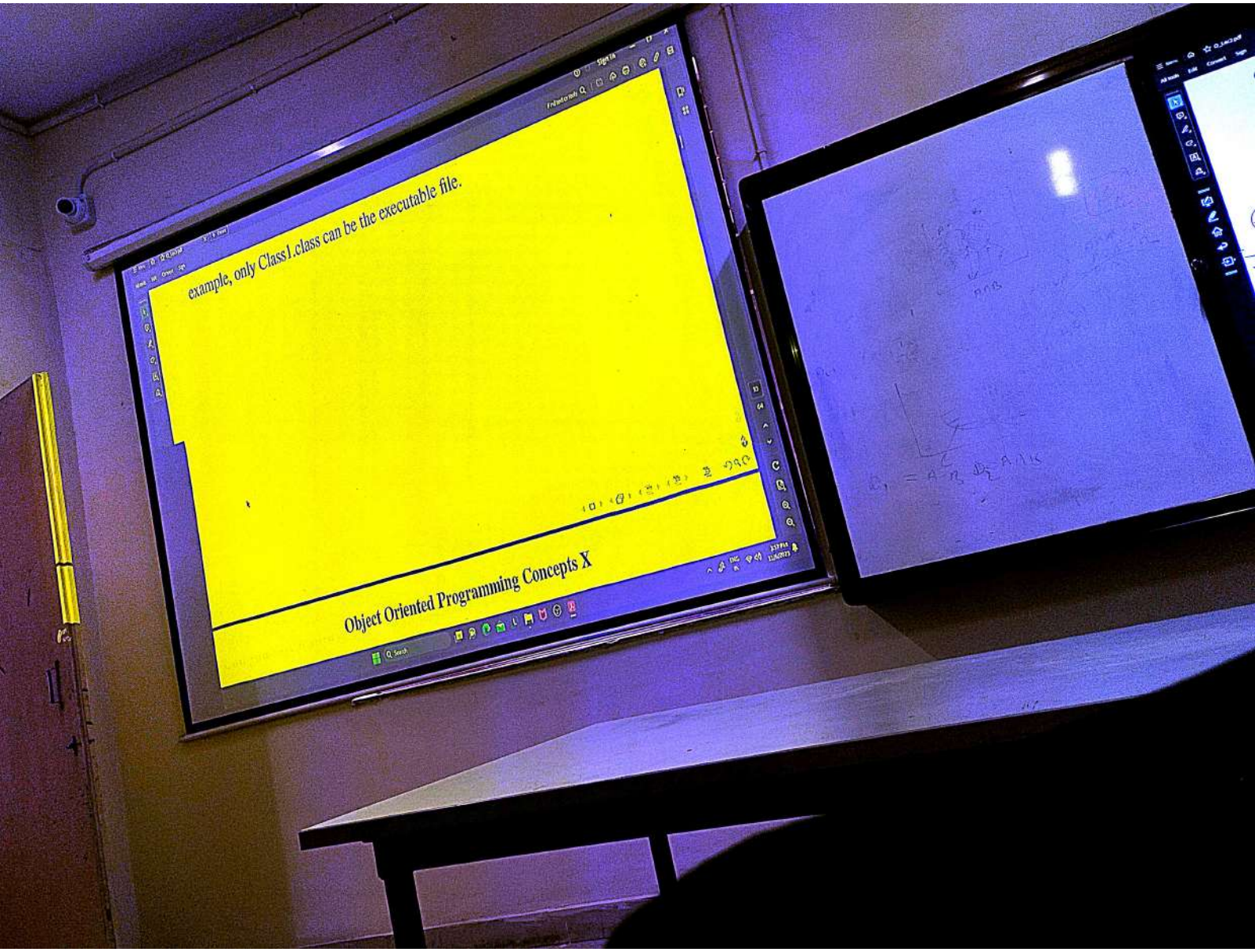

Object Oriented Programming Concepts VI
• Java file name be the same as the public class name that contains main method. Java file name must be "Simple.Java".
public class simple
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
• main method name must be within the public class.

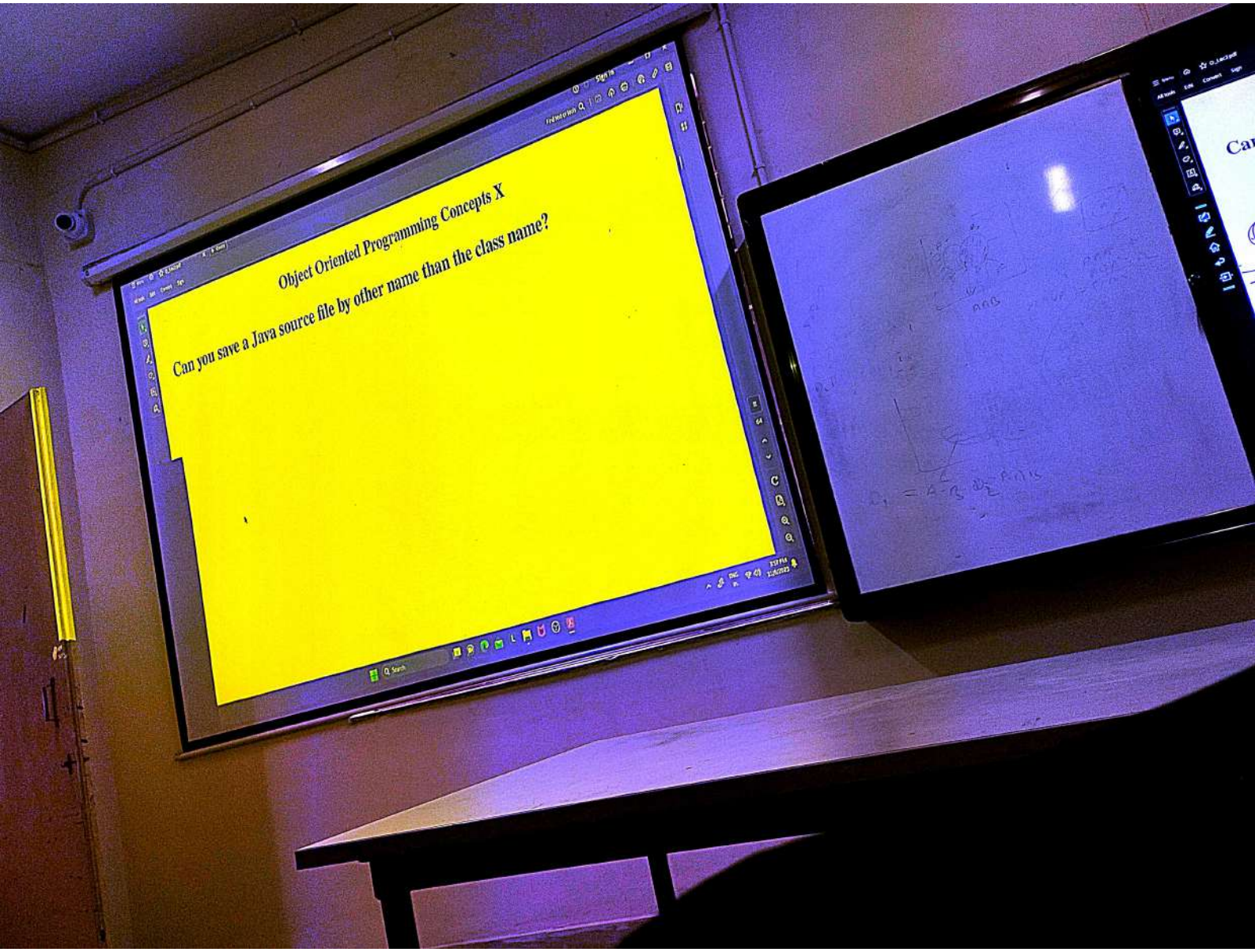

Object Oriented Programming Concepts VII
• One Java file can consist of multiple classes with the restriction that only one of them can be public.
public class Class1
{
}
class Class2
{
}
class Class3
{
}

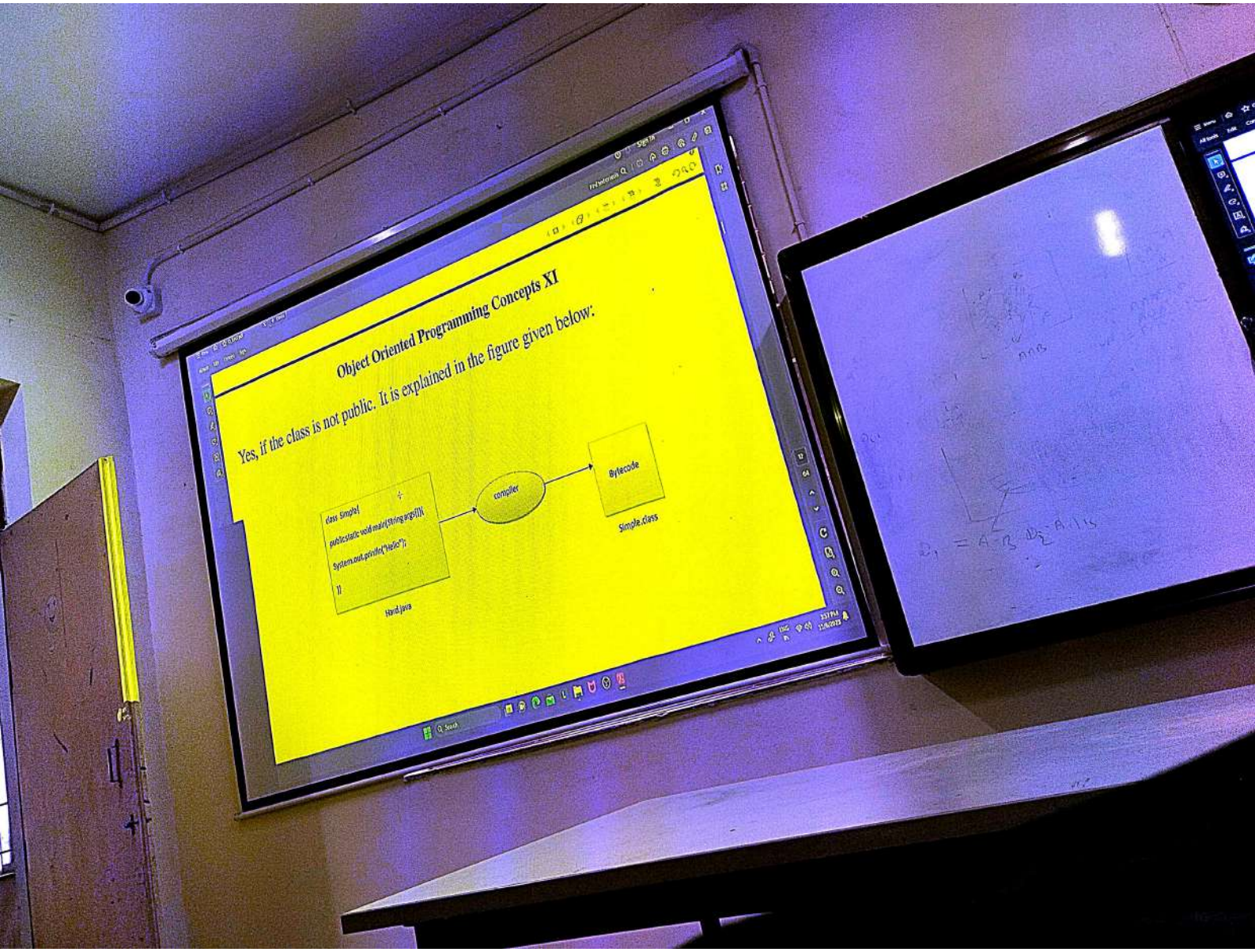

Object Oriented Programming Concepts VIII
• Compiler generates separate .class file of all classes after compilation of Java file.
public class Class1
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
class Class2
{
}
class Class3
{
}
After compilation of Class1.Java, it generates three class files, also called bytecode,

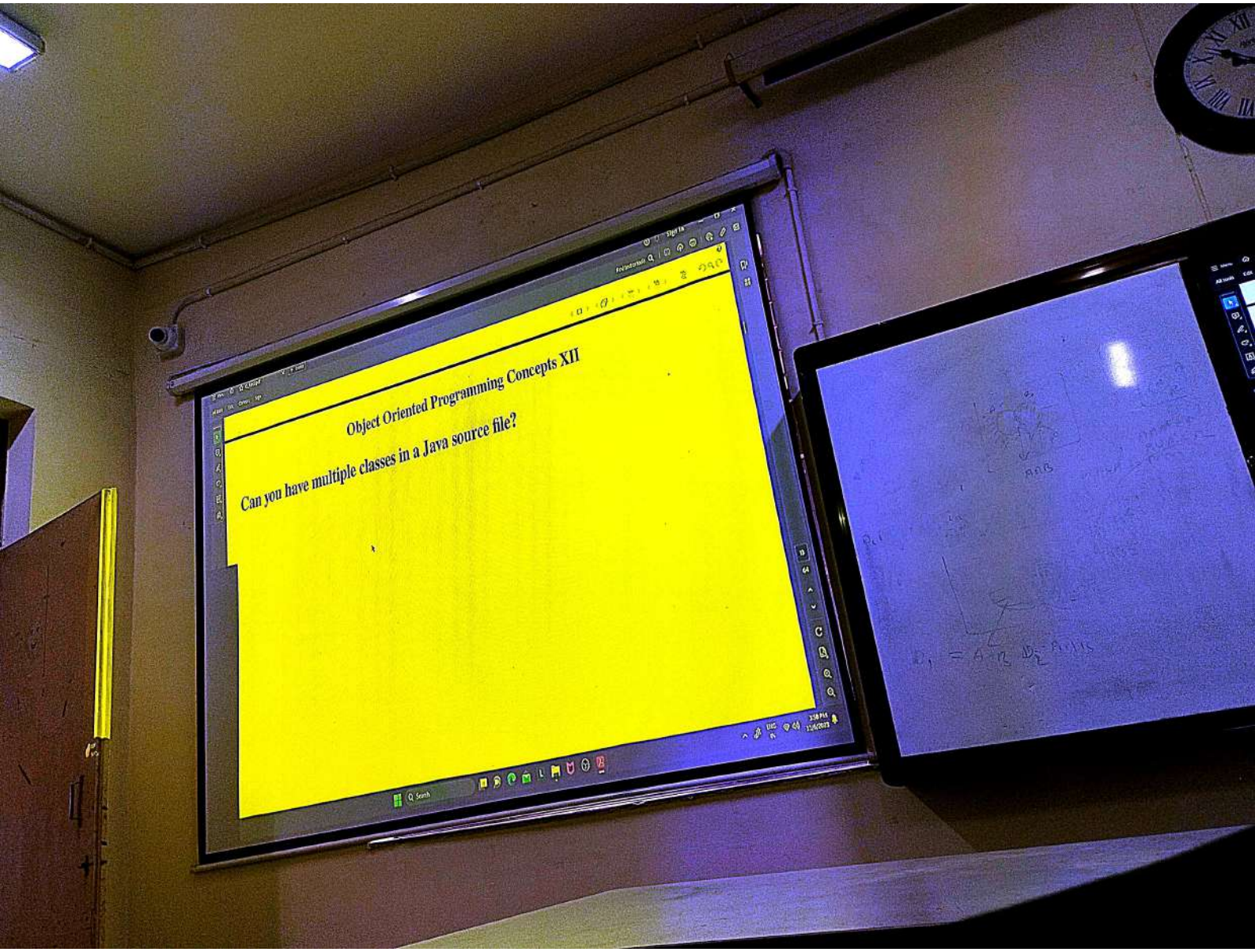

Object Oriented Programming Concepts VIII
• Compiler generates separate .class file of all classes after compilation of Java file.
public class Class1
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
class Class2
{
}
class Class3
{
}
After compilation of Class1.Java, it generates three class files, also called bytecode,
Class1.class, Class2.class and Class3.class.

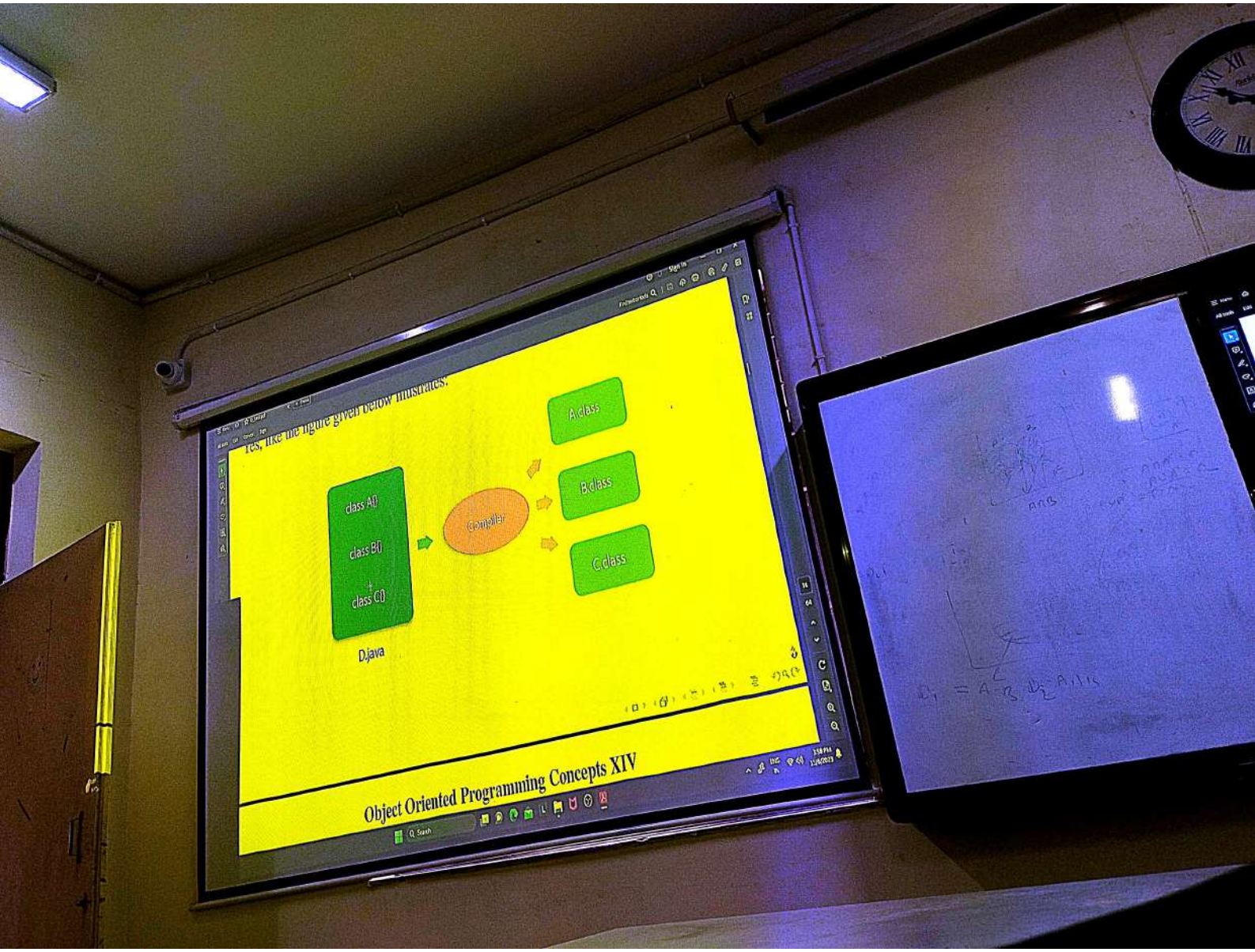

class Class3
{
}
After compilation of Class1.Java, it generates three class files, also called bytecode, Class1.class, Class2.class and Class3.class.
Object Oriented Programming Concepts IX
A class file that contains the main method can be used for execution. In the above example, only Class1.class can be the executable file.

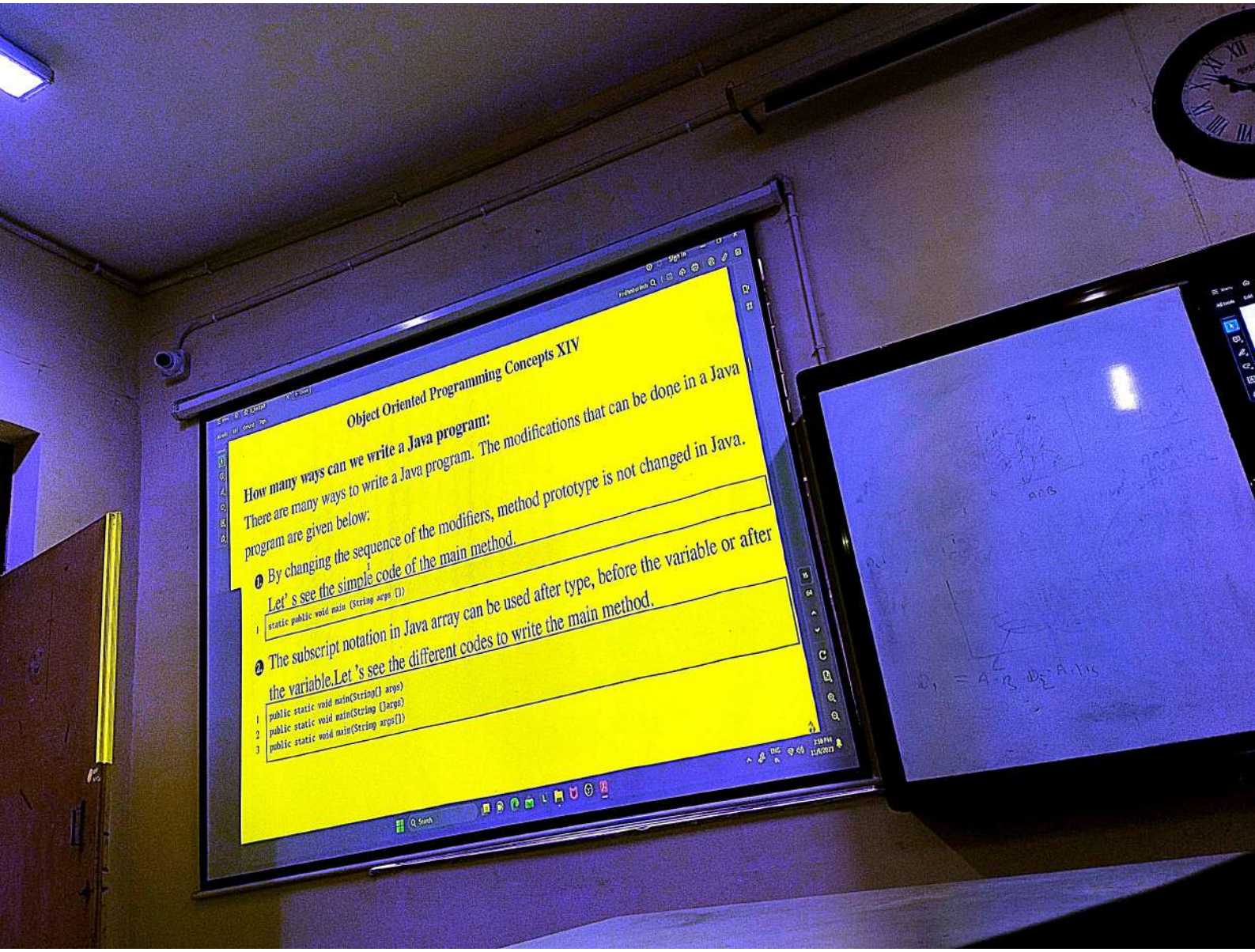

example, only Class1.class can be the executable file.
Object Oriented Programming Concepts X

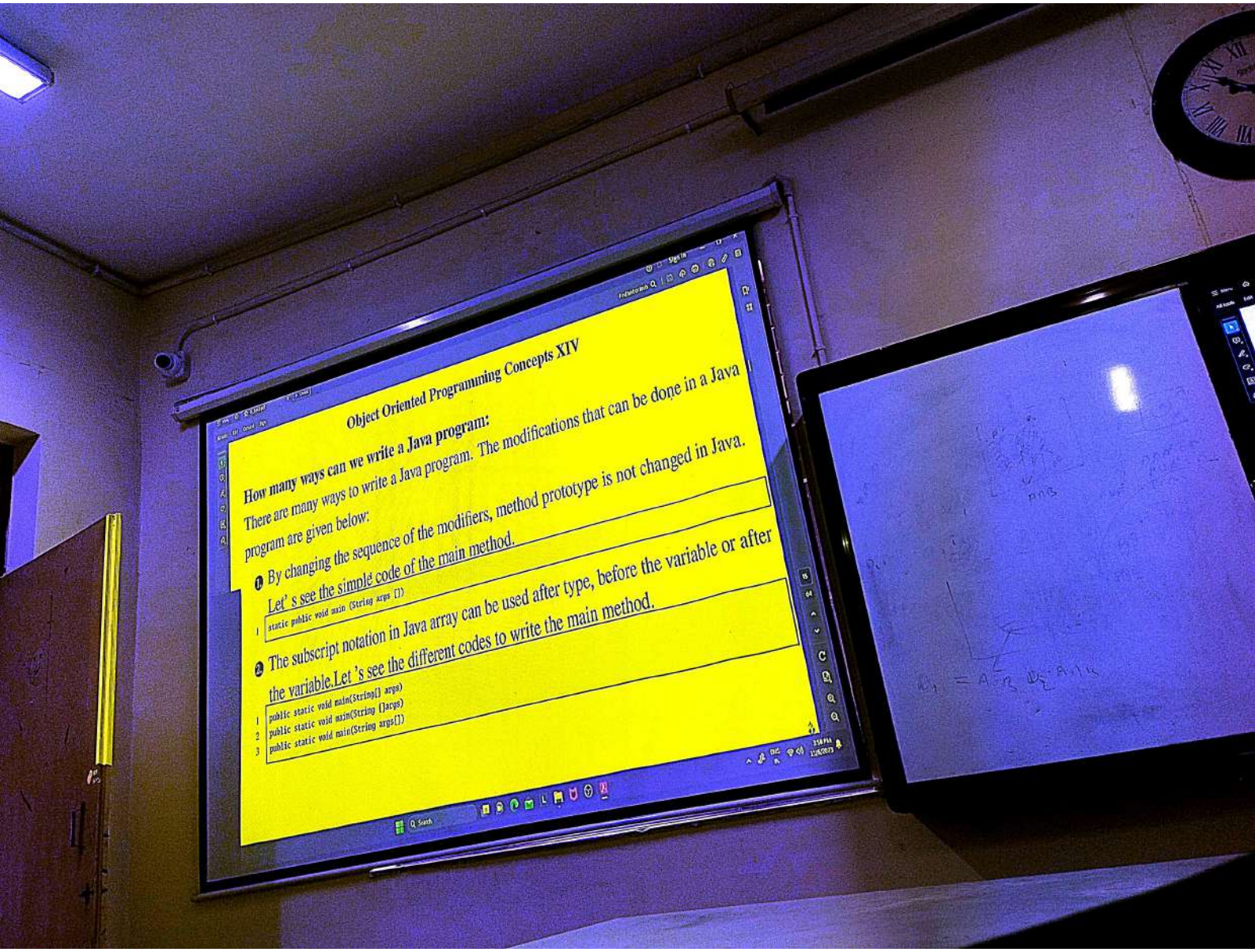

Object Oriented Programming Concepts X
Can you save a Java source file by other name than the class name?

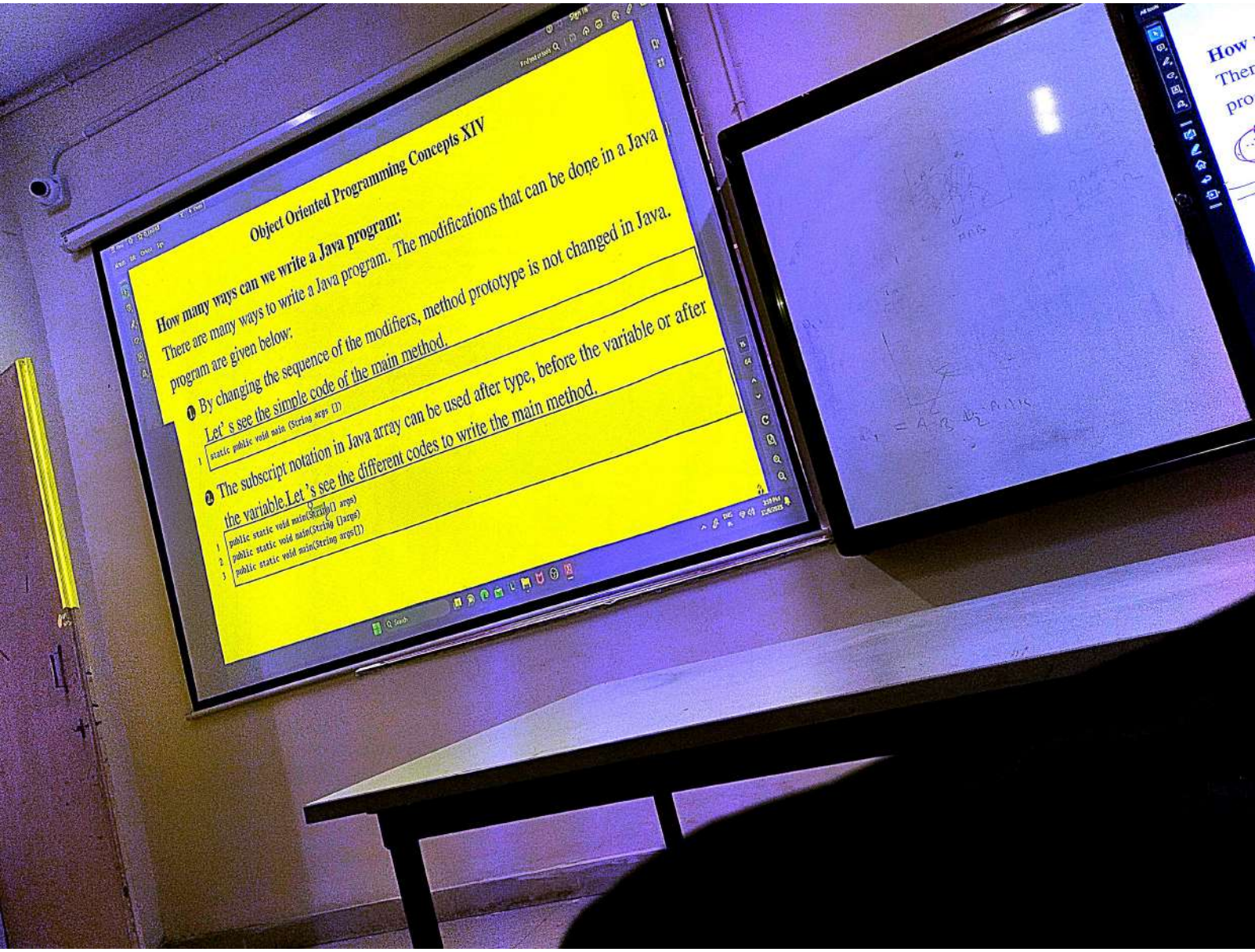

Object Oriented Programming Concepts XI
Yes, if the class is not public. It is explained in the figure given below:
class Simple{
public static void main(String args[]){
System.out.println("Hello");
}}
Hand.java
compiler
Bytecode
Simple.class


Object Oriented Programming Concepts XII
Can you have multiple classes in a Java source file?

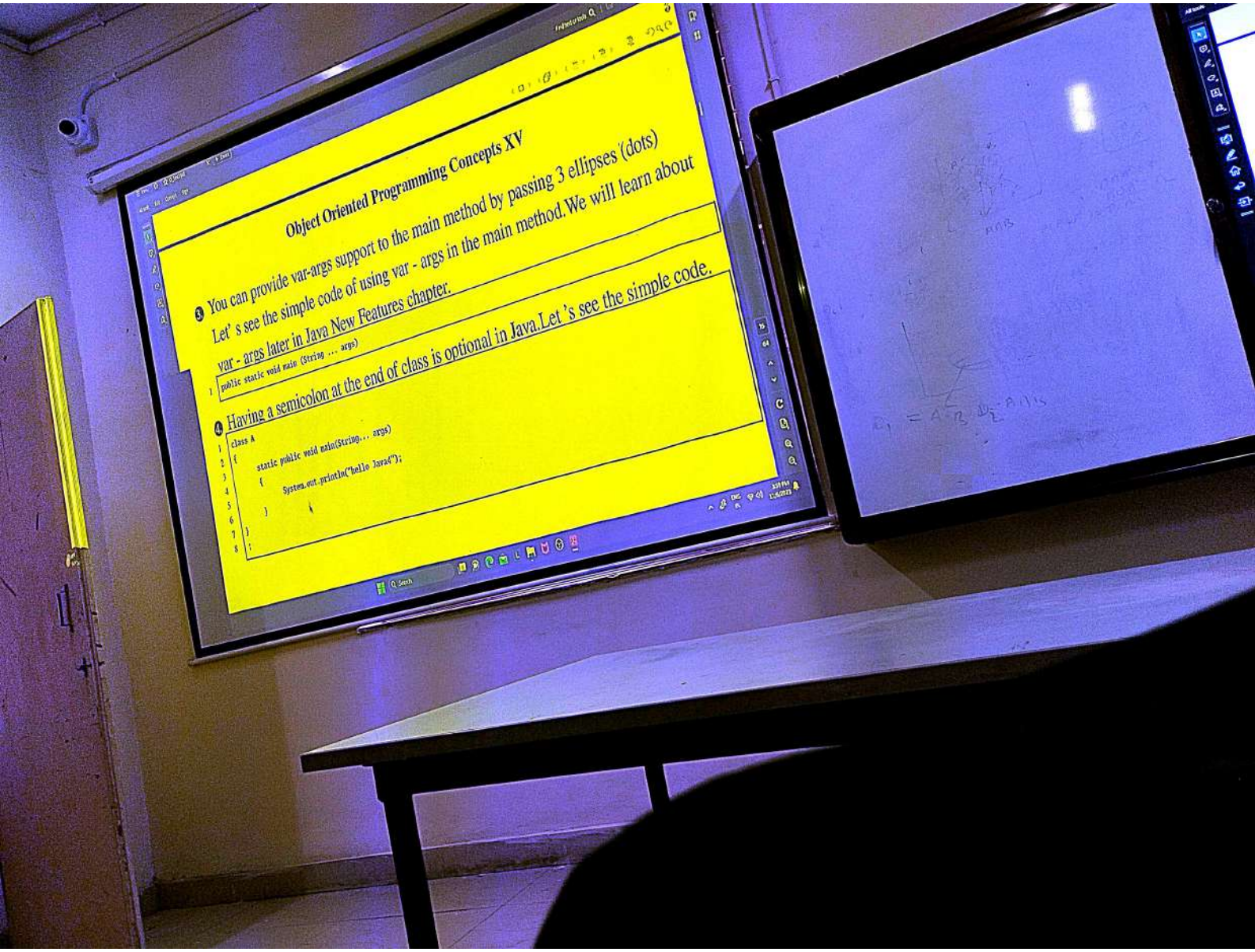

Yes, like the figure given below illustrates:
class A{}
class B{}
class C{}
D.java
Compiler
A.class
B.class
C.class
Object Oriented Programming Concepts XIV


Object Oriented Programming Concepts XIV
How many ways can we write a Java program:
There are many ways to write a Java program. The modifications that can be done in a Java program are given below:
1 By changing the sequence of the modifiers, method prototype is not changed in Java.
Let' s see the simple code of the main method.
1 static public void main (String args [])
2 The subscript notation in Java array can be used after type, before the variable or after the variable.Let 's see the different codes to write the main method.
1 public static void main(String[] args)
2 public static void main(String []args)
3 public static void main(String args[])

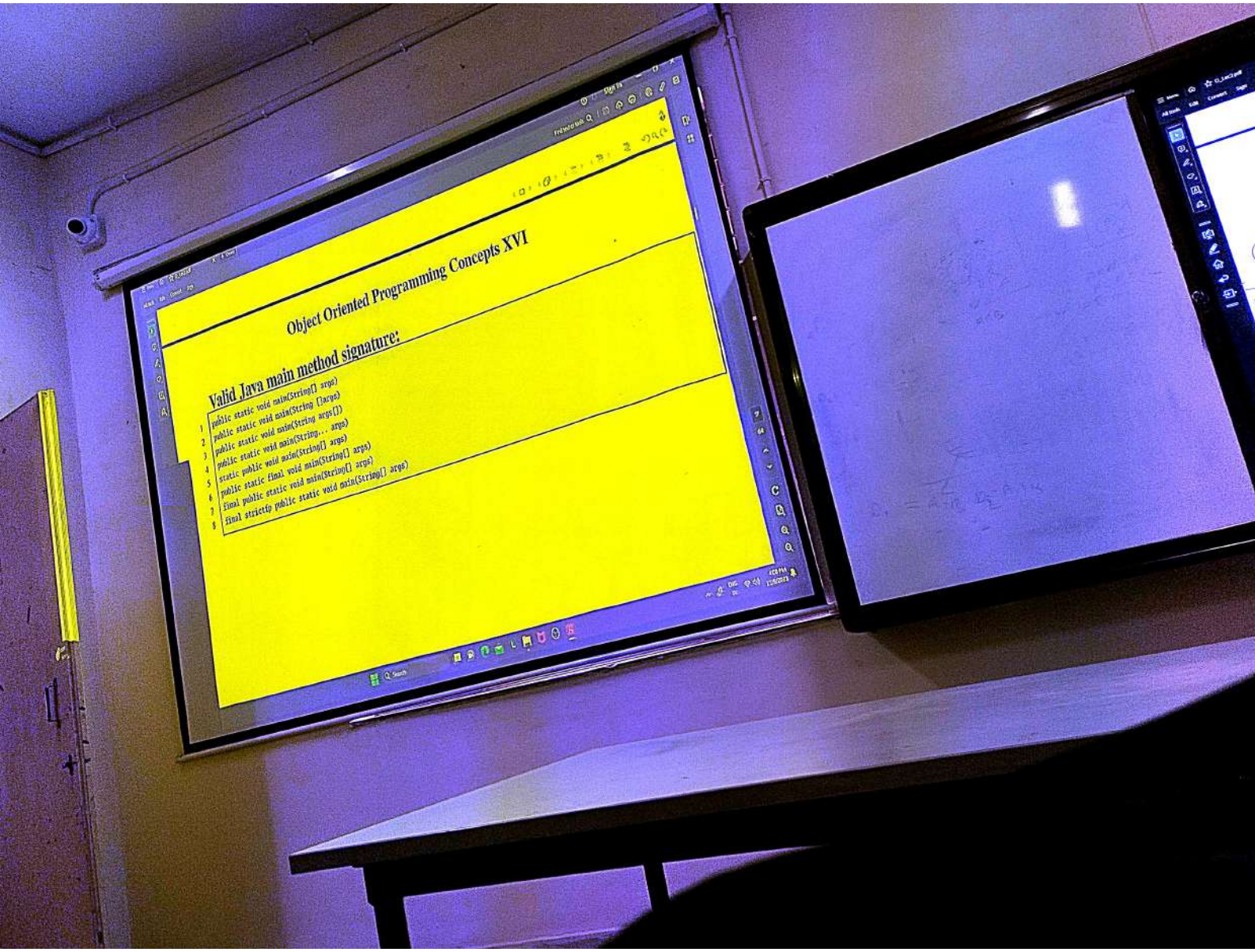

Object Oriented Programming Concepts XIV
How many ways can we write a Java program:
There are many ways to write a Java program. The modifications that can be done in a Java program are given below:
1. By changing the sequence of the modifiers, method prototype is not changed in Java.
Let' s see the simple code of the main method.
static public void main (String args [])
2. The subscript notation in Java array can be used after type, before the variable or after the variable.Let 's see the different codes to write the main method.
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])

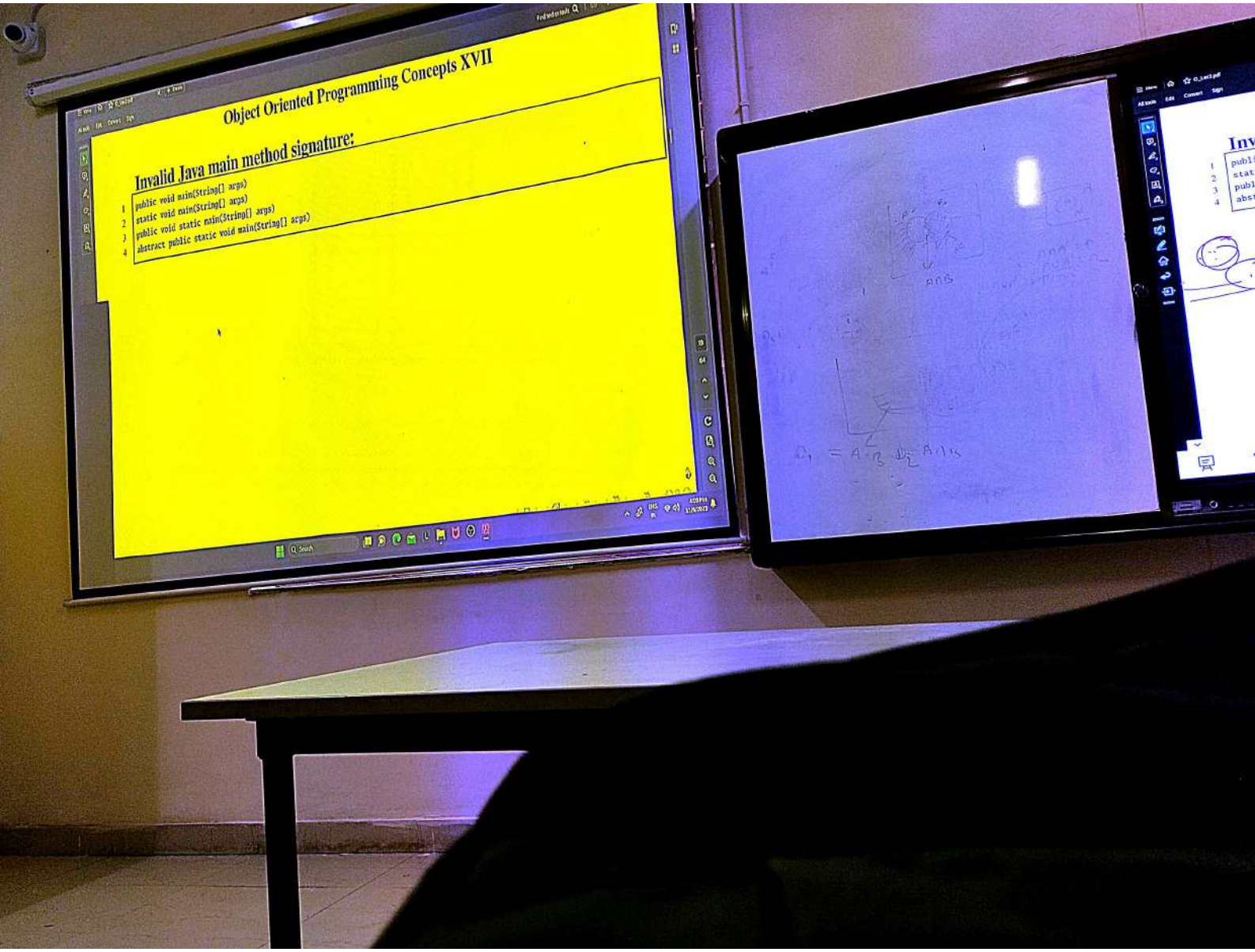

Object Oriented Programming Concepts XIV
How many ways can we write a Java program:
There are many ways to write a Java program. The modifications that can be done in a Java program are given below:
1. By changing the sequence of the modifiers, method prototype is not changed in Java.
Let's see the simple code of the main method.
static public void main (String args [])
2. The subscript notation in Java array can be used after type, before the variable or after the variable.Let's see the different codes to write the main method.
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])

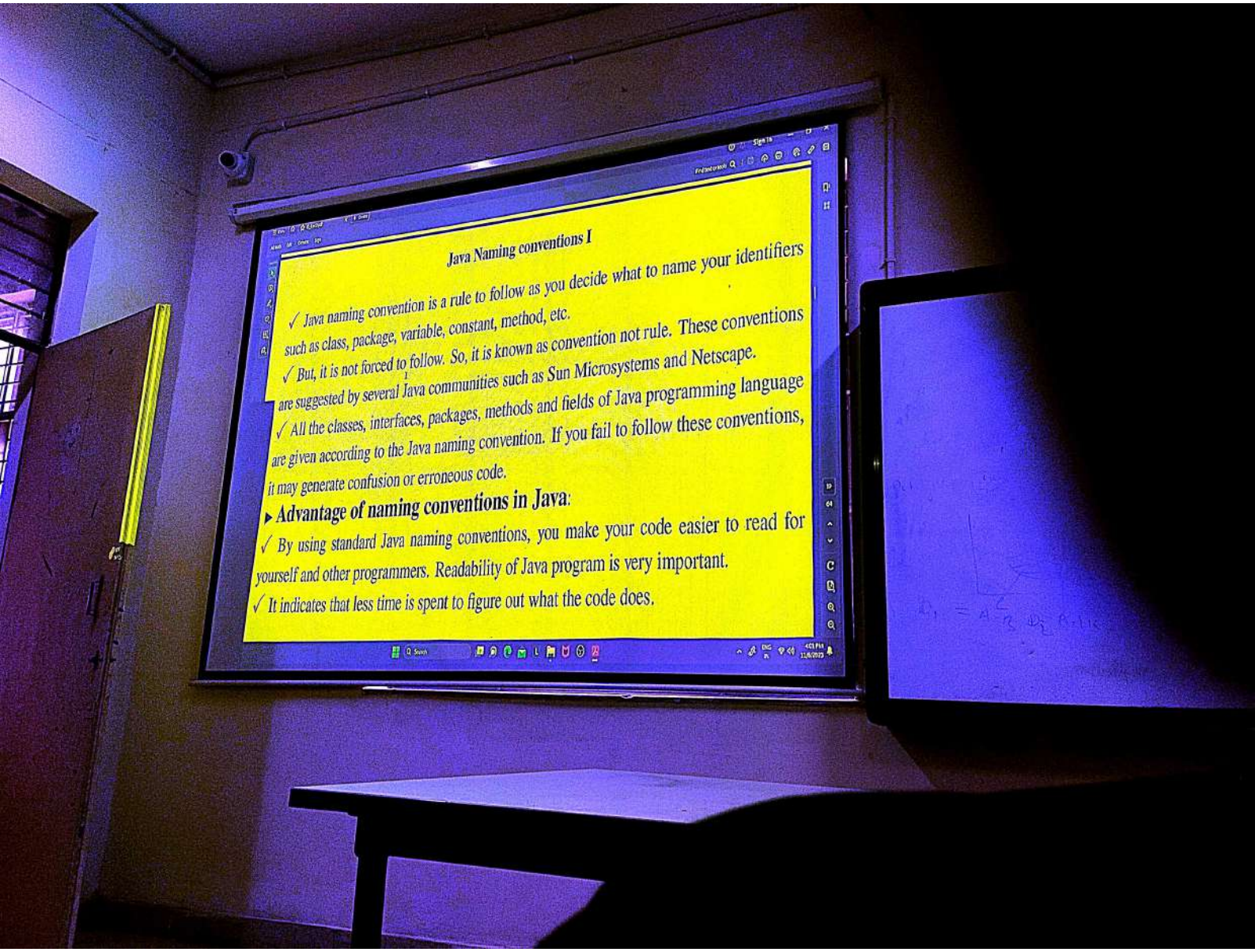

Object Oriented Programming Concepts XIV
How many ways can we write a Java program:
There are many ways to write a Java program. The modifications that can be done in a Java program are given below:
By changing the sequence of the modifiers, method prototype is not changed in Java.
Let' s see the simple code of the main method.
1 static public void main (String args [])
The subscript notation in Java array can be used after type, before the variable or after the variable.Let 's see the different codes to write the main method.
1 public static void main(String[] args)
2 public static void main(String[]args)
3 public static void main(String args[])

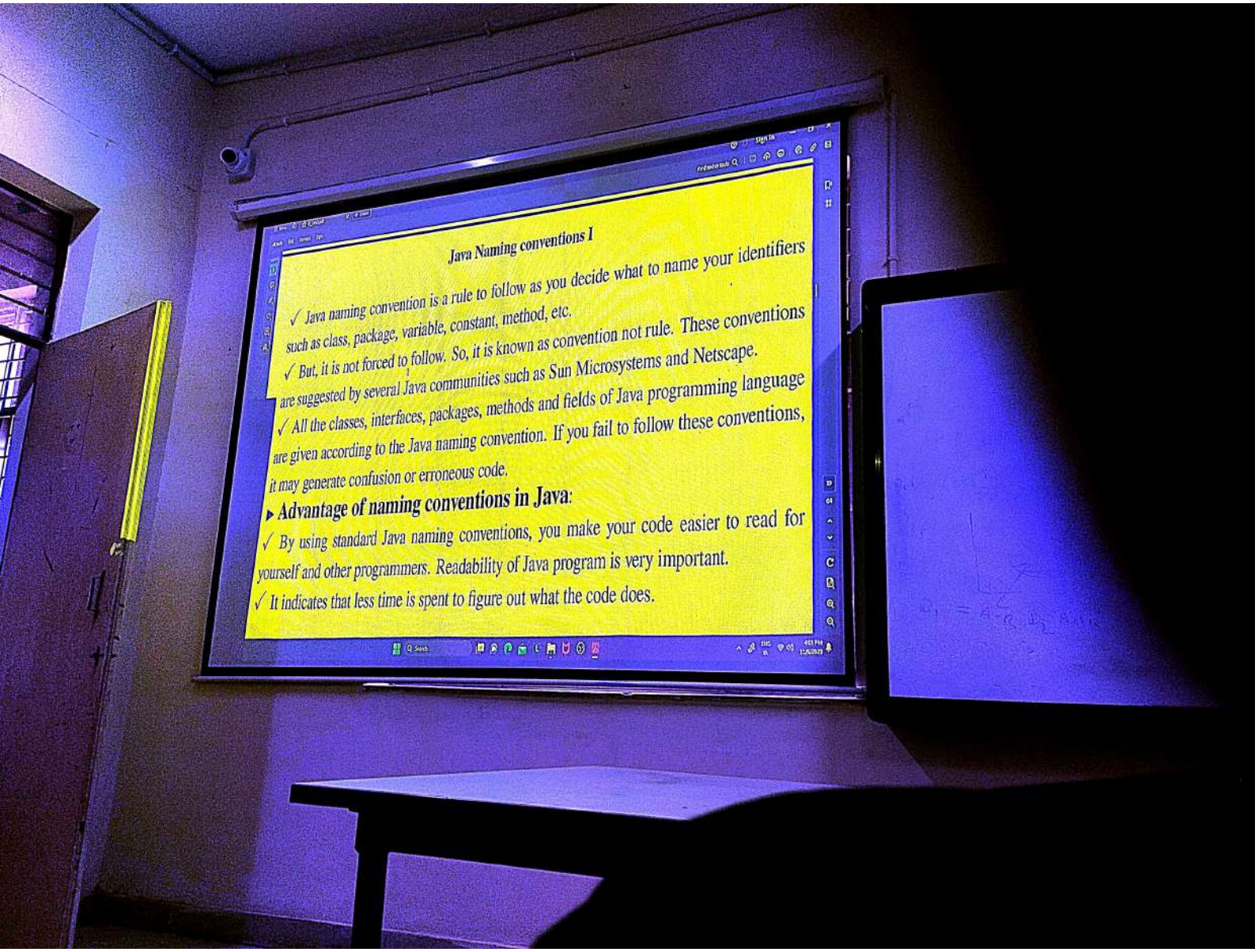

Object Oriented Programming Concepts XV
You can provide var-args support to the main method by passing 3 ellipses (dots)
Let's see the simple code of using var - args in the main method.We will learn about var - args later in Java New Features chapter.
public static void main (String ... args)
Having a semicolon at the end of class is optional in Java.Let's see the simple code.
class A
{
static public void main(String... args)
{
System.out.println("Hello Java4");
}
};

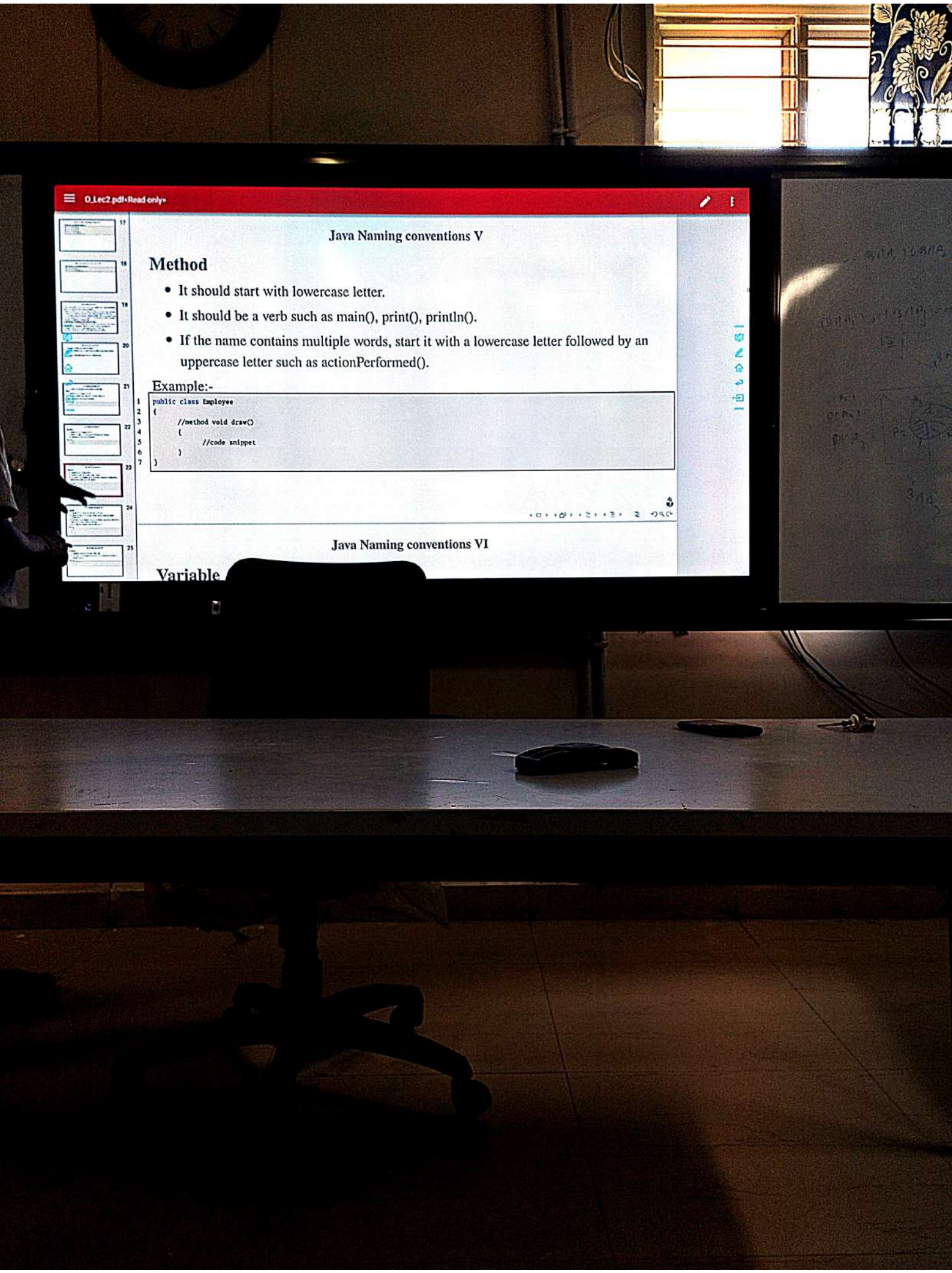

Object Oriented Programming Concepts XVI
Valid Java main method signature:
1 public static void main(String[] args)
2 public static void main(String []args)
3 public static void main(String... args)
4 public static void main(String[] args)
5 static public final void main(String[] args)
6 public static final void main(String[] args)
7 final public static void main(String[] args)
8 final strictfp public static void main(String[] args)

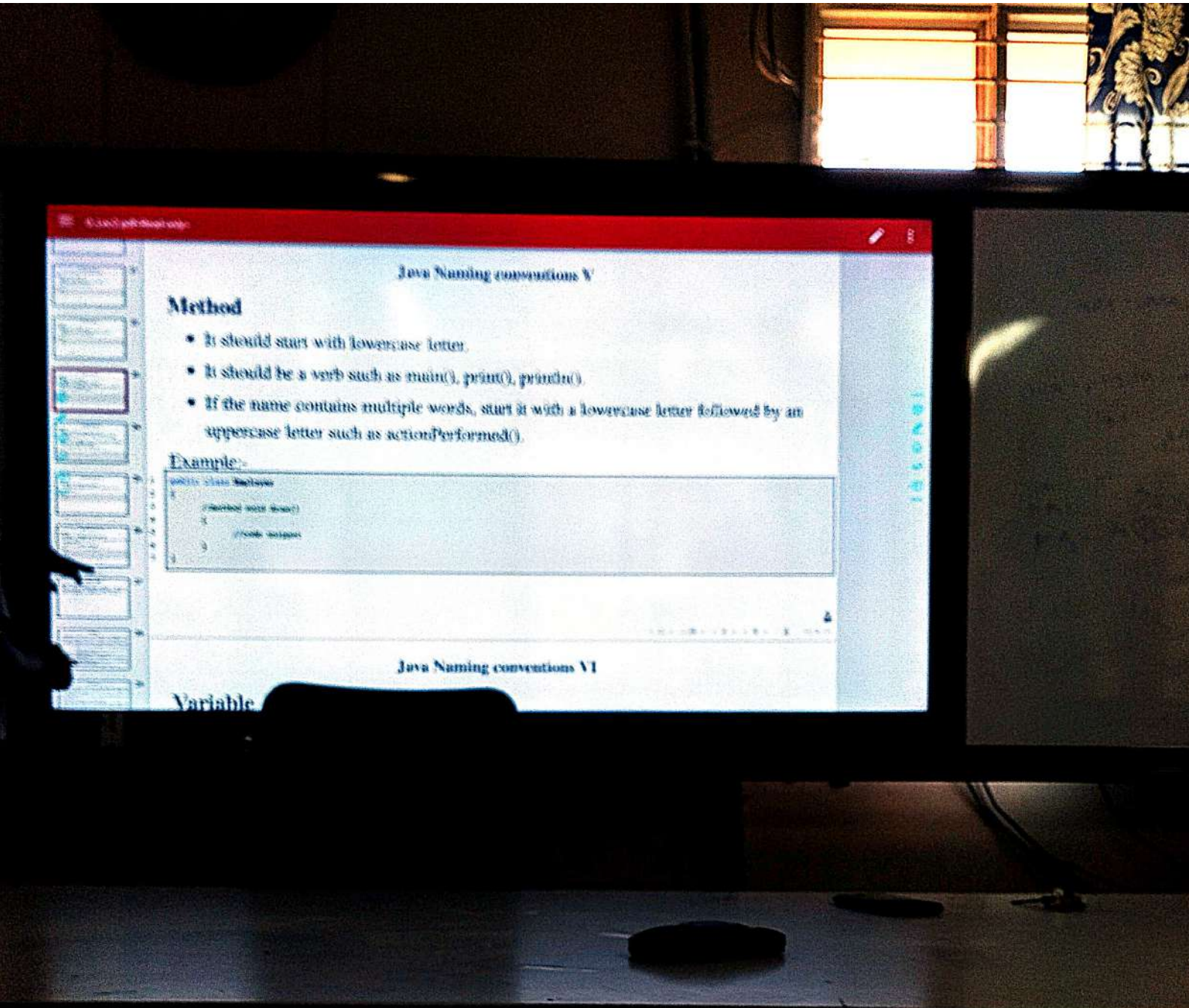

Object Oriented Programming Concepts XVI
Valid Java main method signature:
1 public static void main(String[] args)
2 public static void main(String []args)
3 public static void main(String args[])
4 public static void main(String... args)
5 static public void main(String[] args)
6 public static final void main(String[] args)
7 final public static void main(String[] args)
8 final strictfp public static void main(String[] args)

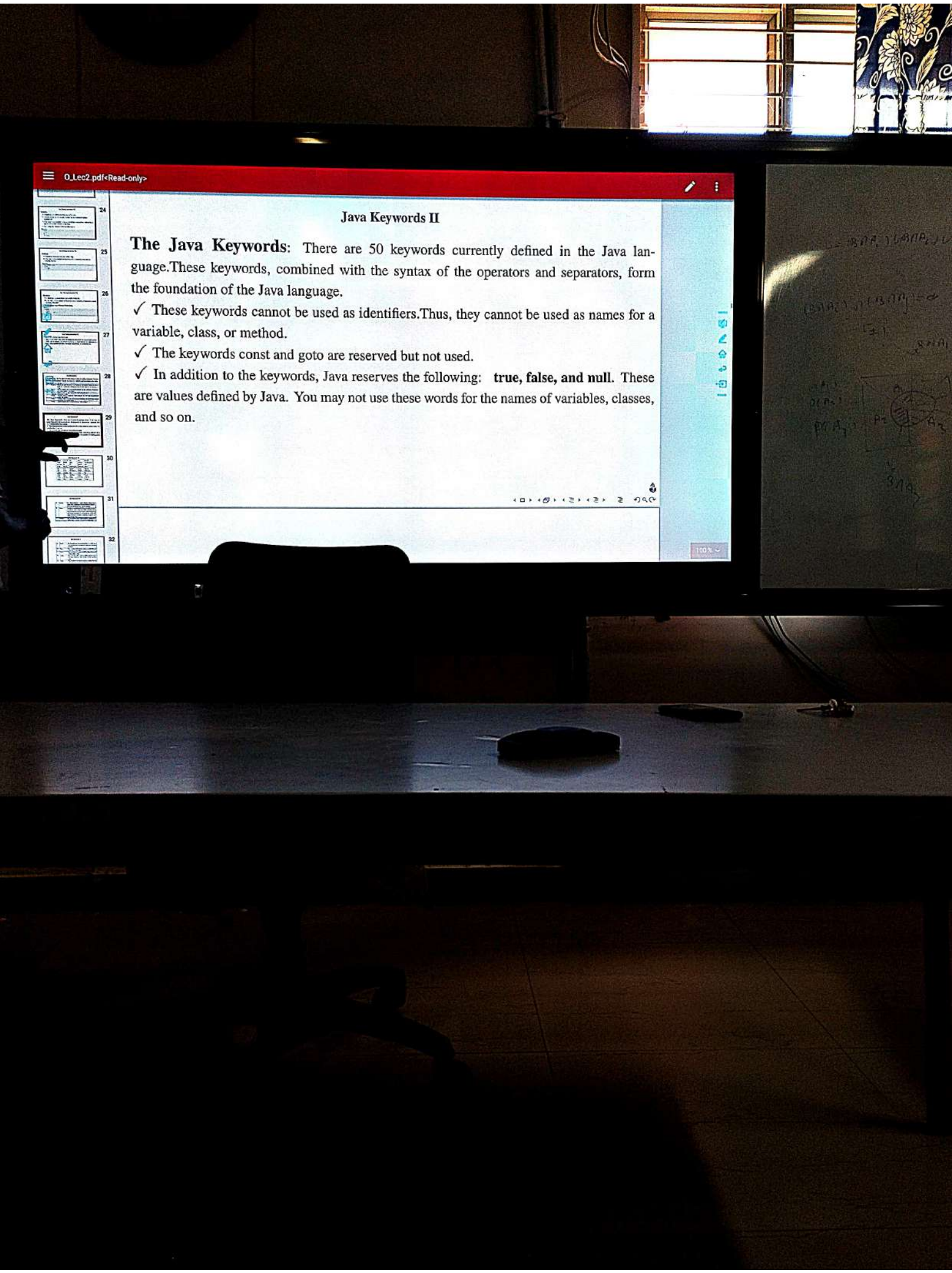

Object Oriented Programming Concepts XVII
Invalid Java main method signature:
1 public void main(String[] args)
2 static void main(String[] args)
3 public void static main(String[] args)
4 abstract public static void main(String[] args)

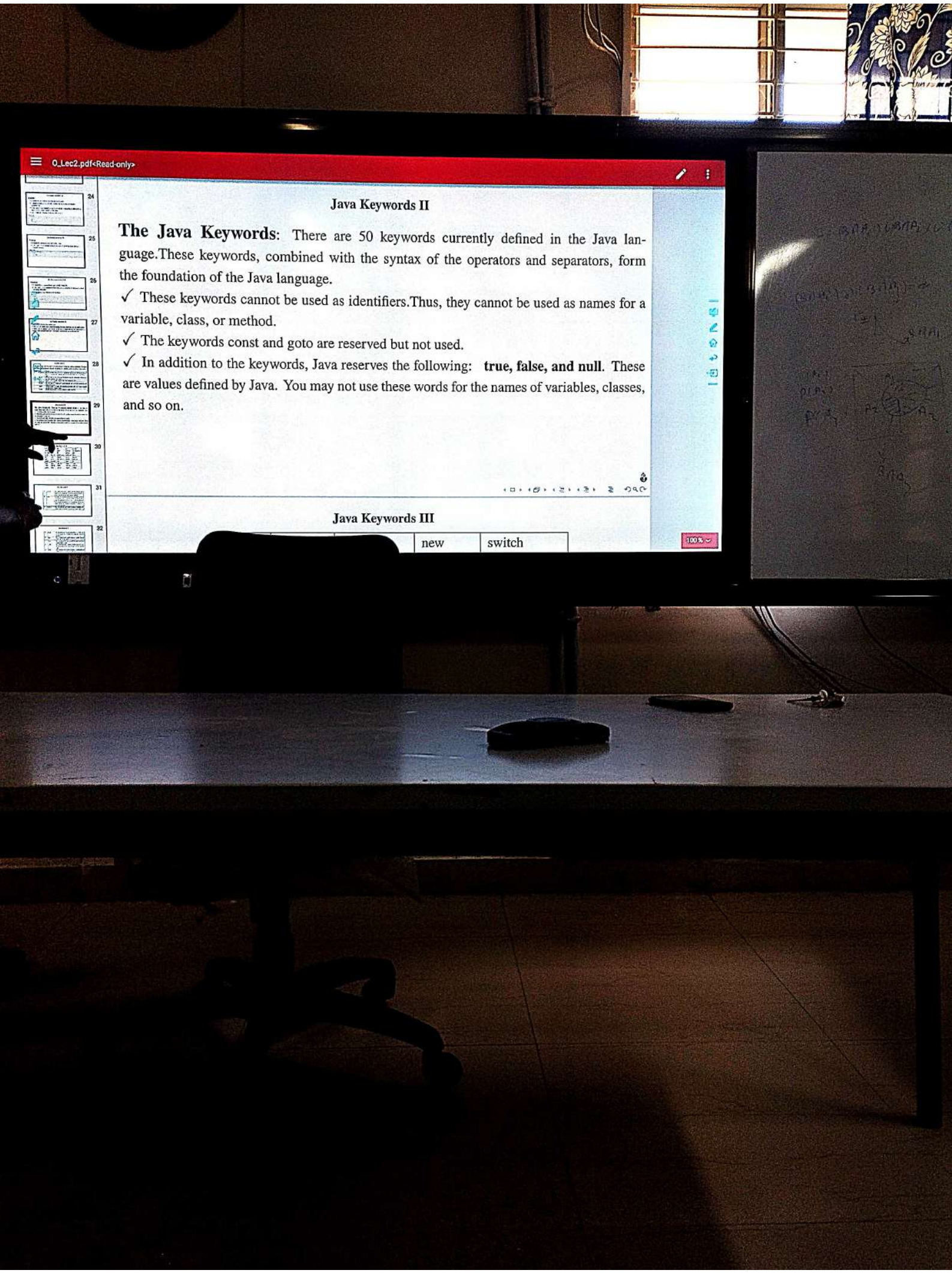

Java Naming conventions I
✓ Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
✓ But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
✓ All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.
▸ Advantage of naming conventions in Java:
✓ By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important.
✓ It indicates that less time is spent to figure out what the code does.

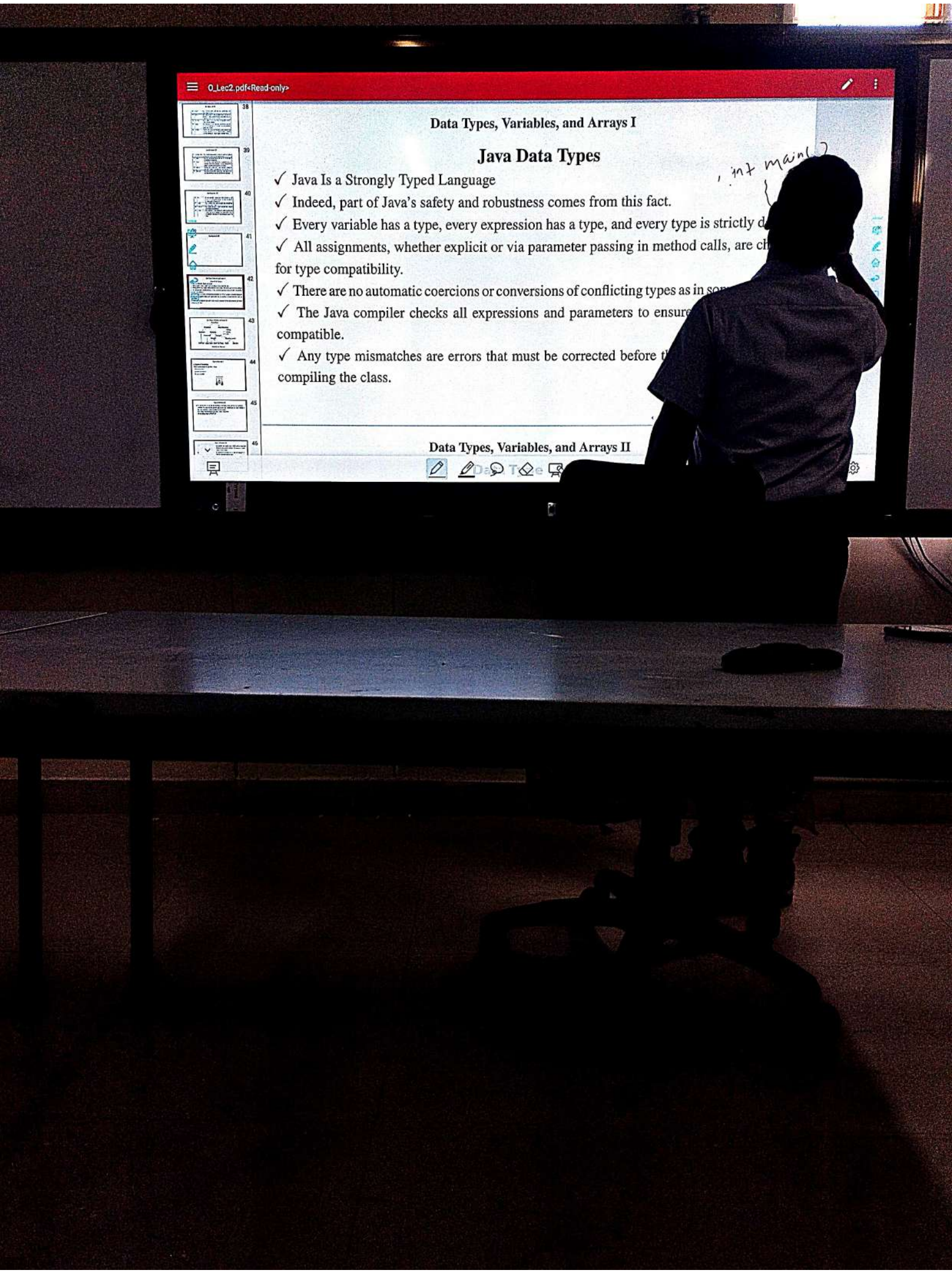

Java Naming conventions I
✓ Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
✓ But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
✓ All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.
▸ Advantage of naming conventions in Java:
✓ By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important.
✓ It indicates that less time is spent to figure out what the code does.

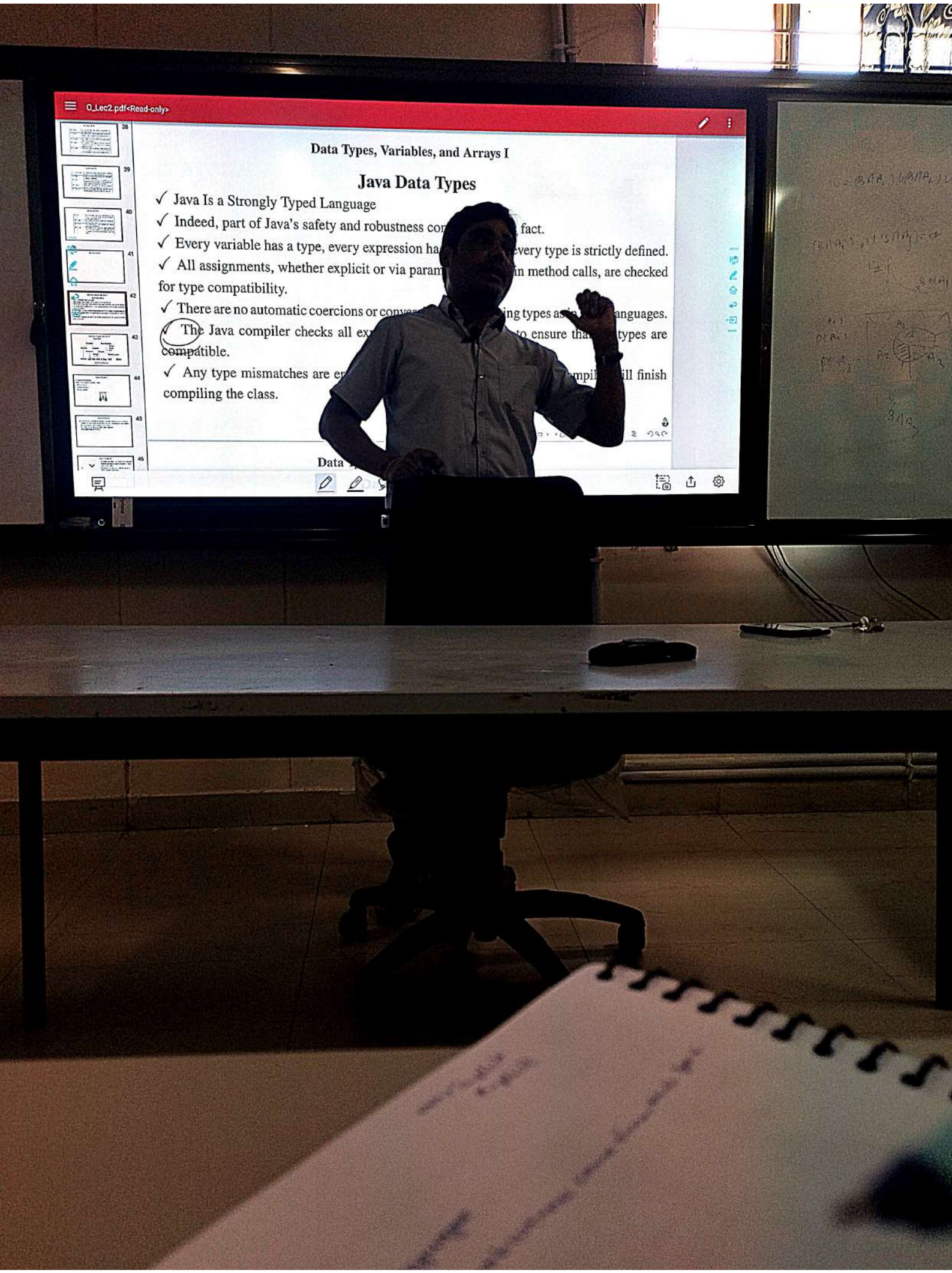

O_Lec2.pdf•Read only•
Java Naming conventions V
Method
It should start with lowercase letter.
It should be a verb such as main(), print(), println().
If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
Example:-
public class Employee
{
//method void draw()
{
//code snippet
}
}
Java Naming conventions VI
Variable

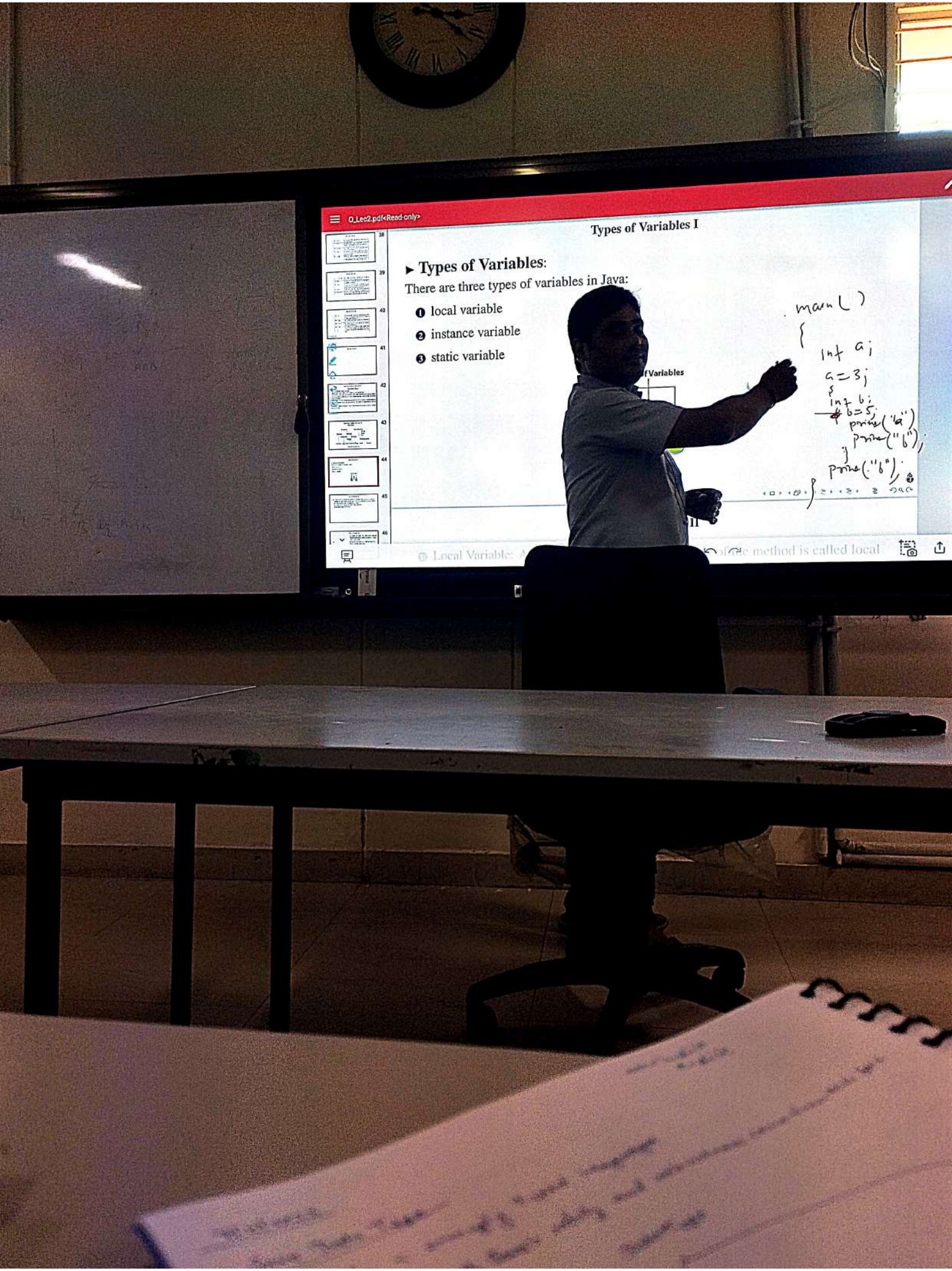

Java Naming conventions V
Method
It should start with lowercase letter.
It should be a verb such as main(), print(), println().
If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
Example:-
Java Naming conventions VI
Variable

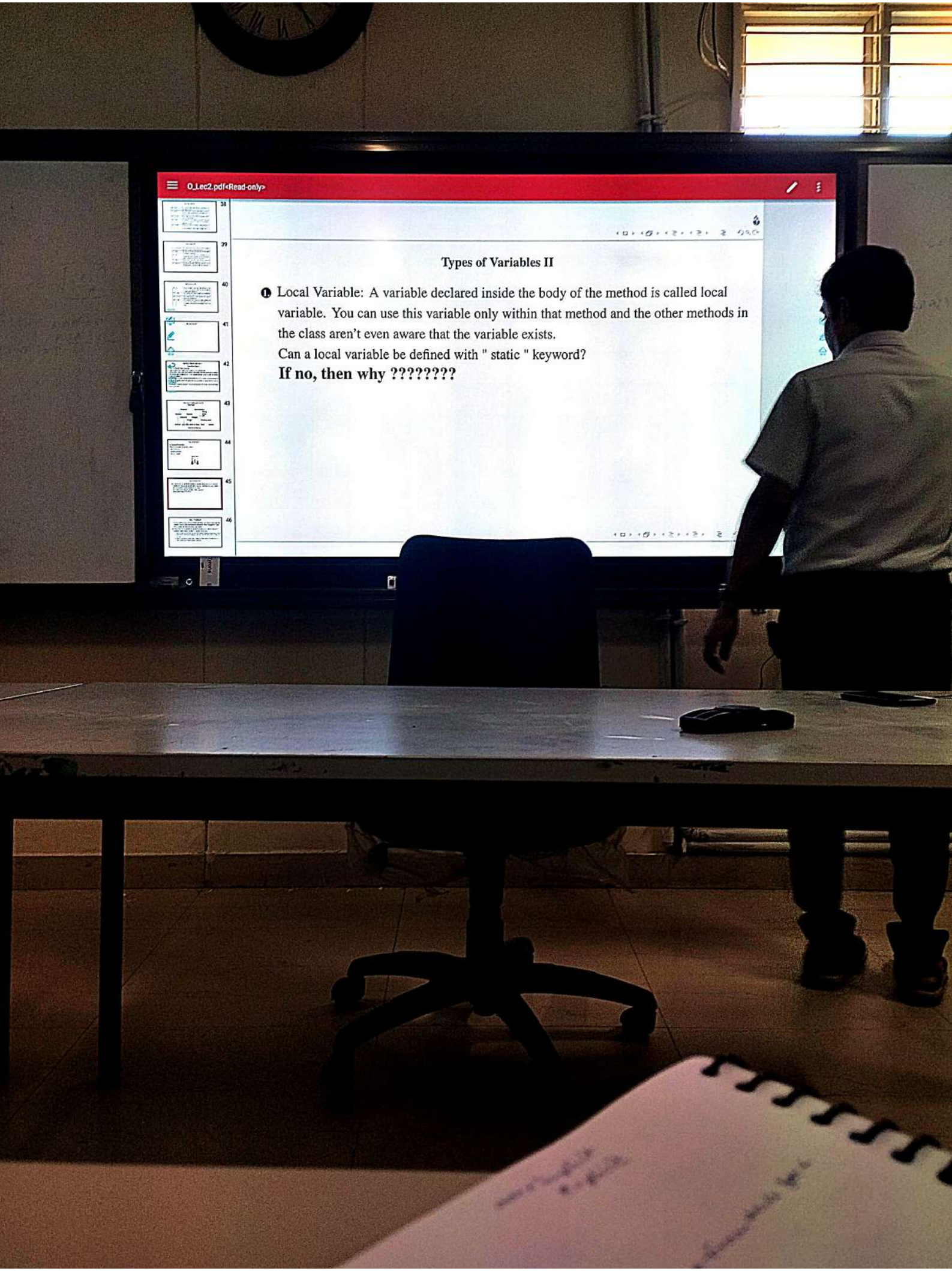

O_Lec2.pdf<Read-only>
Java Keywords II
The Java Keywords: There are 50 keywords currently defined in the Java language.These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
✓ These keywords cannot be used as identifiers.Thus, they cannot be used as names for a variable, class, or method.
✓ The keywords const and goto are reserved but not used.
✓ In addition to the keywords, Java reserves the following: true, false, and null. These are values defined by Java. You may not use these words for the names of variables, classes, and so on.

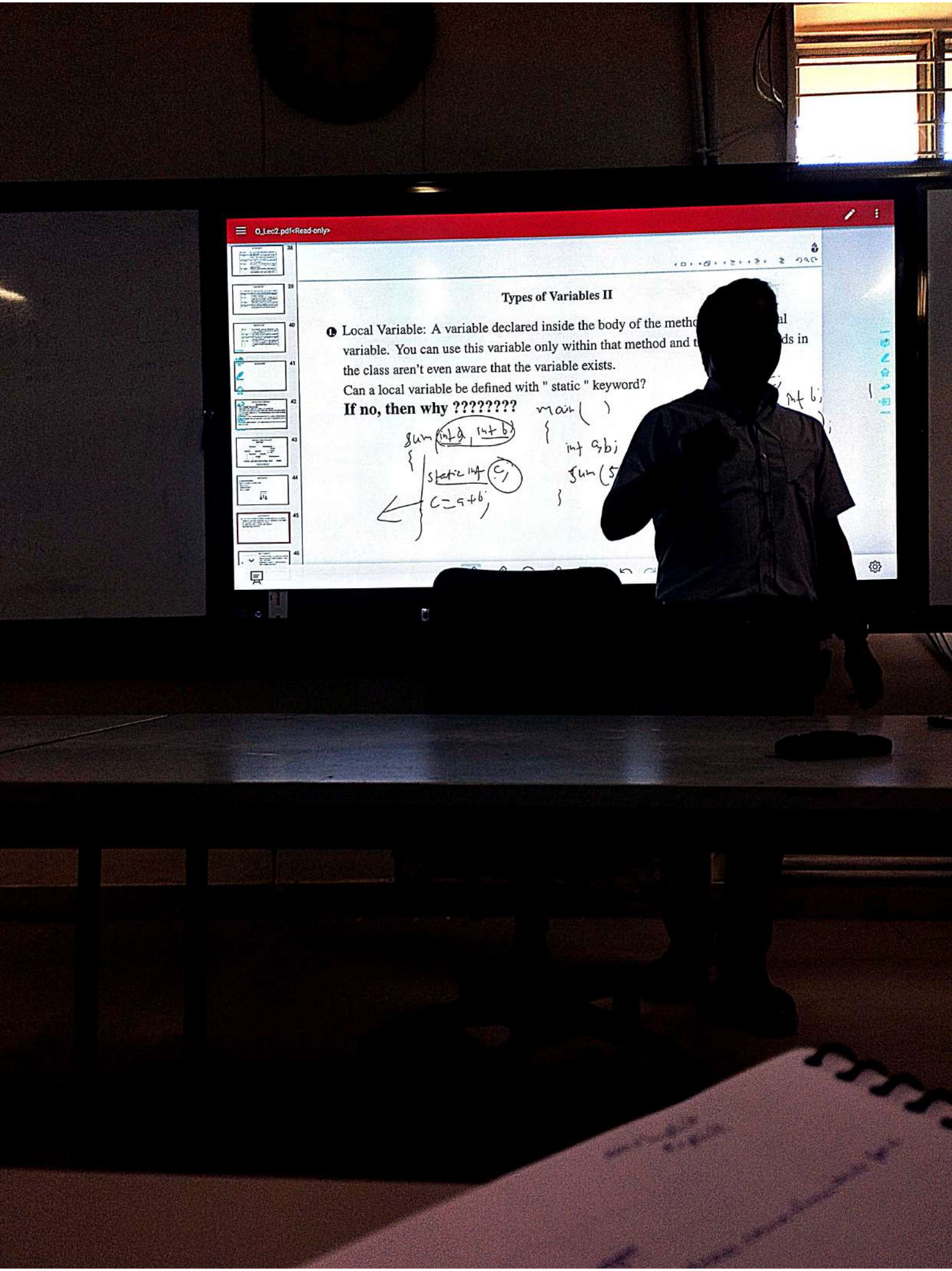

O_Lec2.pdf<Read-only>
Java Keywords II
The Java Keywords: There are 50 keywords currently defined in the Java language.These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
✓ These keywords cannot be used as identifiers.Thus, they cannot be used as names for a variable, class, or method.
✓ The keywords const and goto are reserved but not used.
✓ In addition to the keywords, Java reserves the following: true, false, and null. These are values defined by Java. You may not use these words for the names of variables, classes, and so on.
Java Keywords III
new
switch
100%

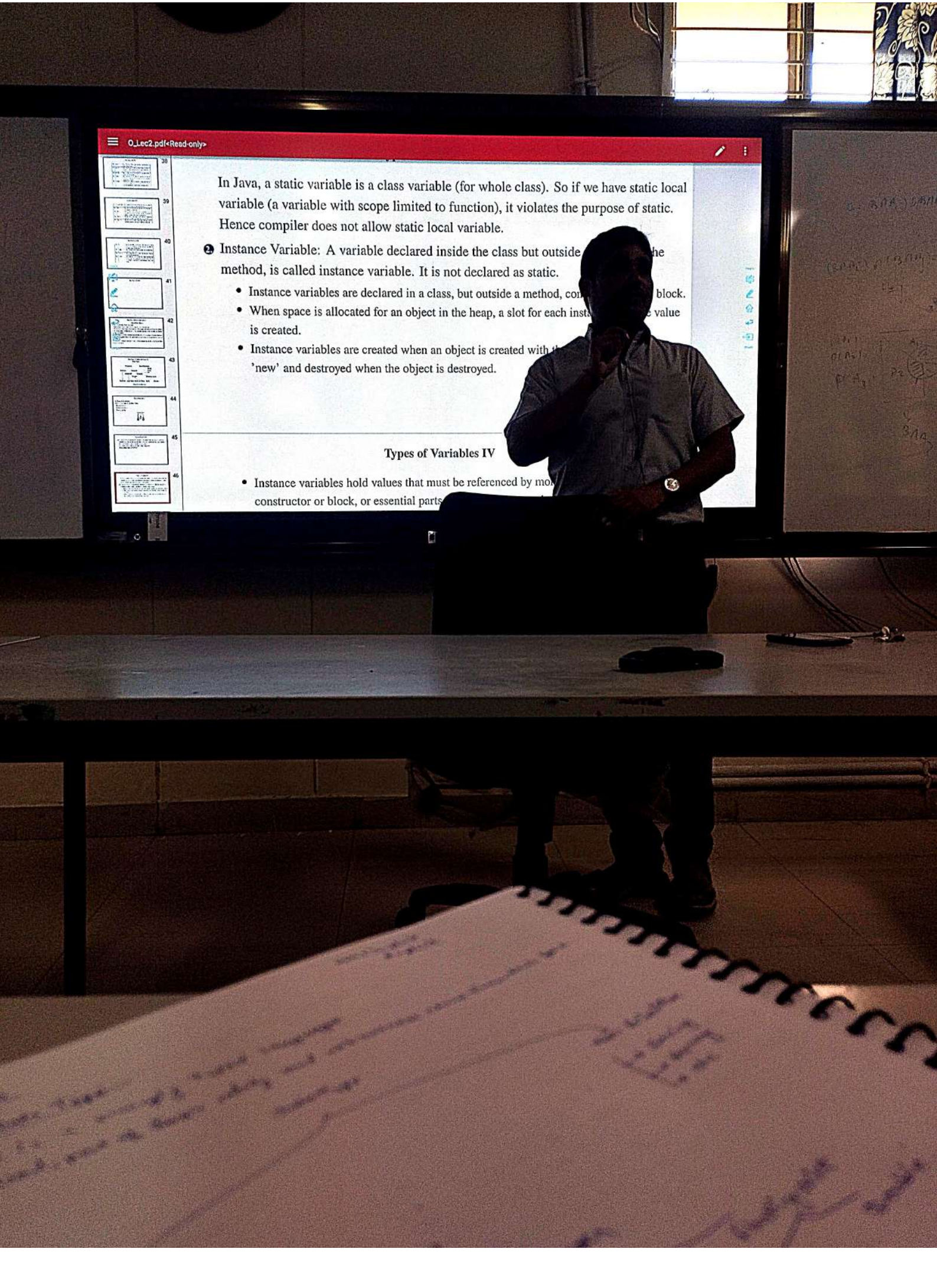

0_Lec2.pdf<Read-only>
Data Types, Variables, and Arrays I
Java Data Types
✓ Java Is a Strongly Typed Language
✓ Indeed, part of Java's safety and robustness comes from this fact.
✓ Every variable has a type, every expression has a type, and every type is strictly d
✓ All assignments, whether explicit or via parameter passing in method calls, are ch
for type compatibility.
✓ There are no automatic coercions or conversions of conflicting types as in so
✓ The Java compiler checks all expressions and parameters to ensure
compatible.
✓ Any type mismatches are errors that must be corrected before t
compiling the class.
Data Types, Variables, and Arrays II
int main()

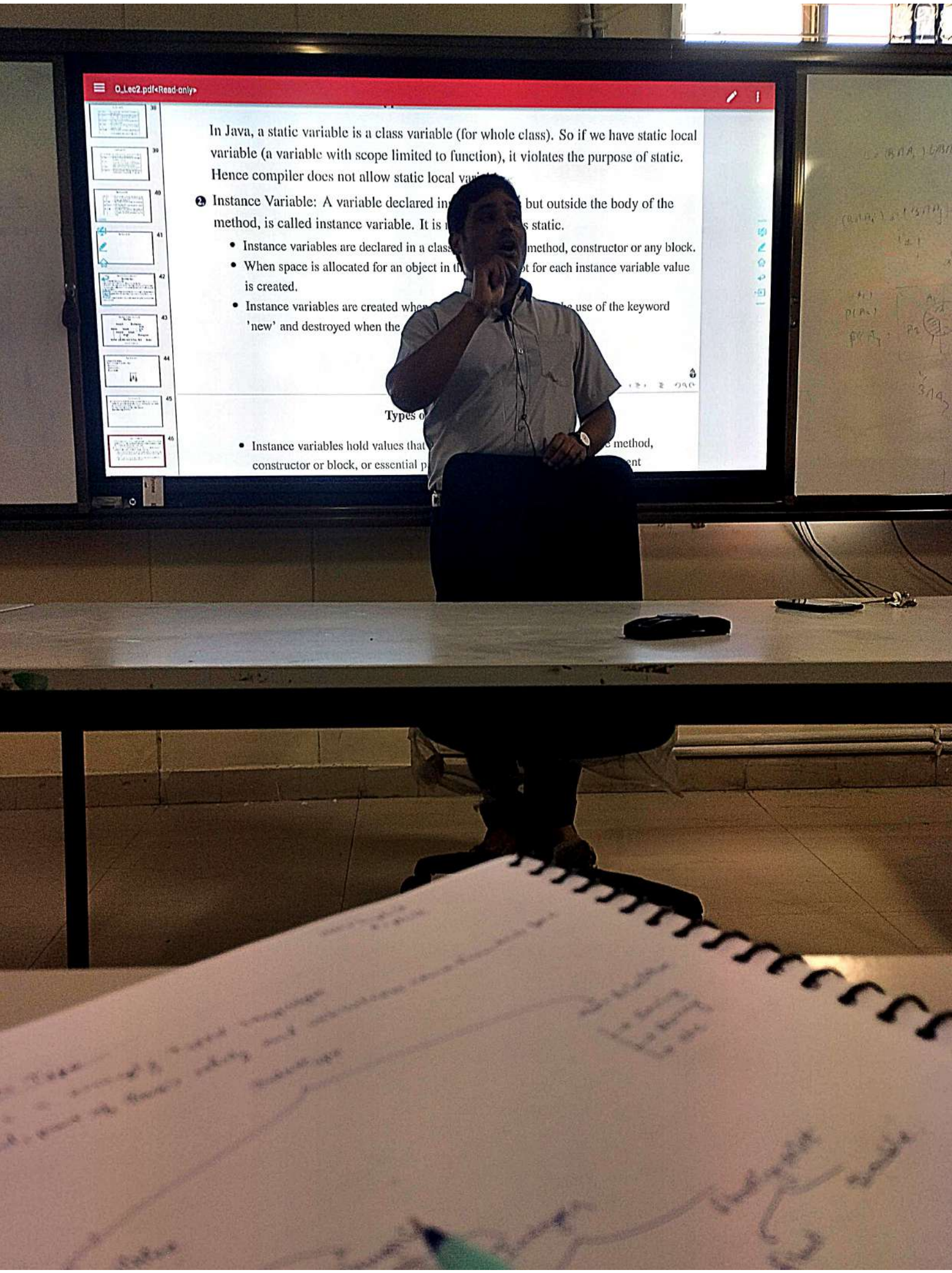

O_Lec2.pdf<Read-only>
Data Types, Variables, and Arrays I
Java Data Types
✓ Java Is a Strongly Typed Language
✓ Indeed, part of Java's safety and robustness co fact.
✓ Every variable has a type, every expression ha very type is strictly defined.
✓ All assignments, whether explicit or via param in method calls, are checked for type compatibility.
✓ There are no automatic coercions or conver ing types as anguages.
✓ The Java compiler checks all ex to ensure tha types are compatible.
✓ Any type mismatches are er mpil ill finish compiling the class.
Data

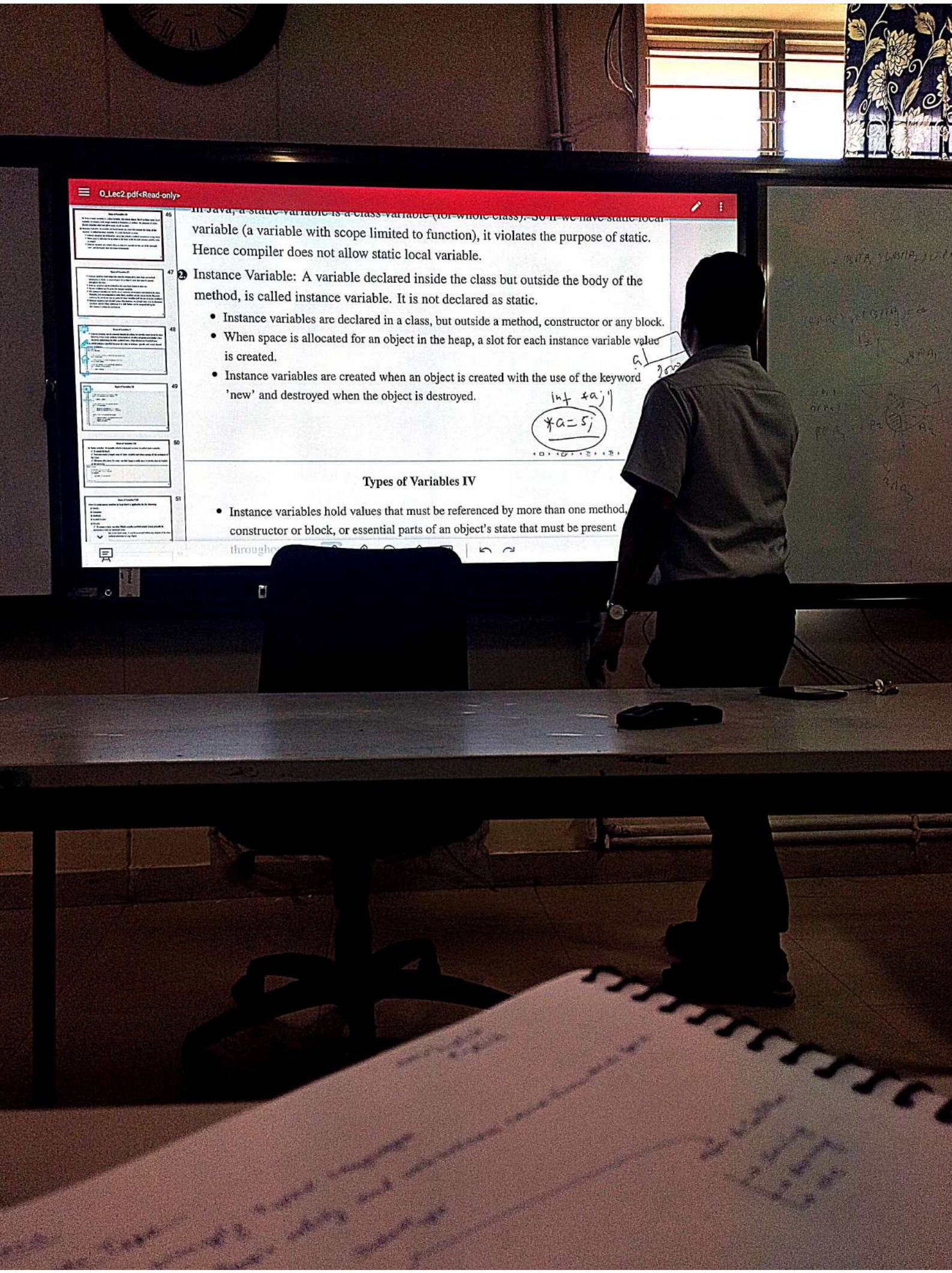

O_Lec2.pdf<Read-only>
Types of Variables I
Types of Variables:
There are three types of variables in Java:
1 local variable
2 instance variable
3 static variable
main()
int a;
a=3;
int b;
b=5;
Local Variable:

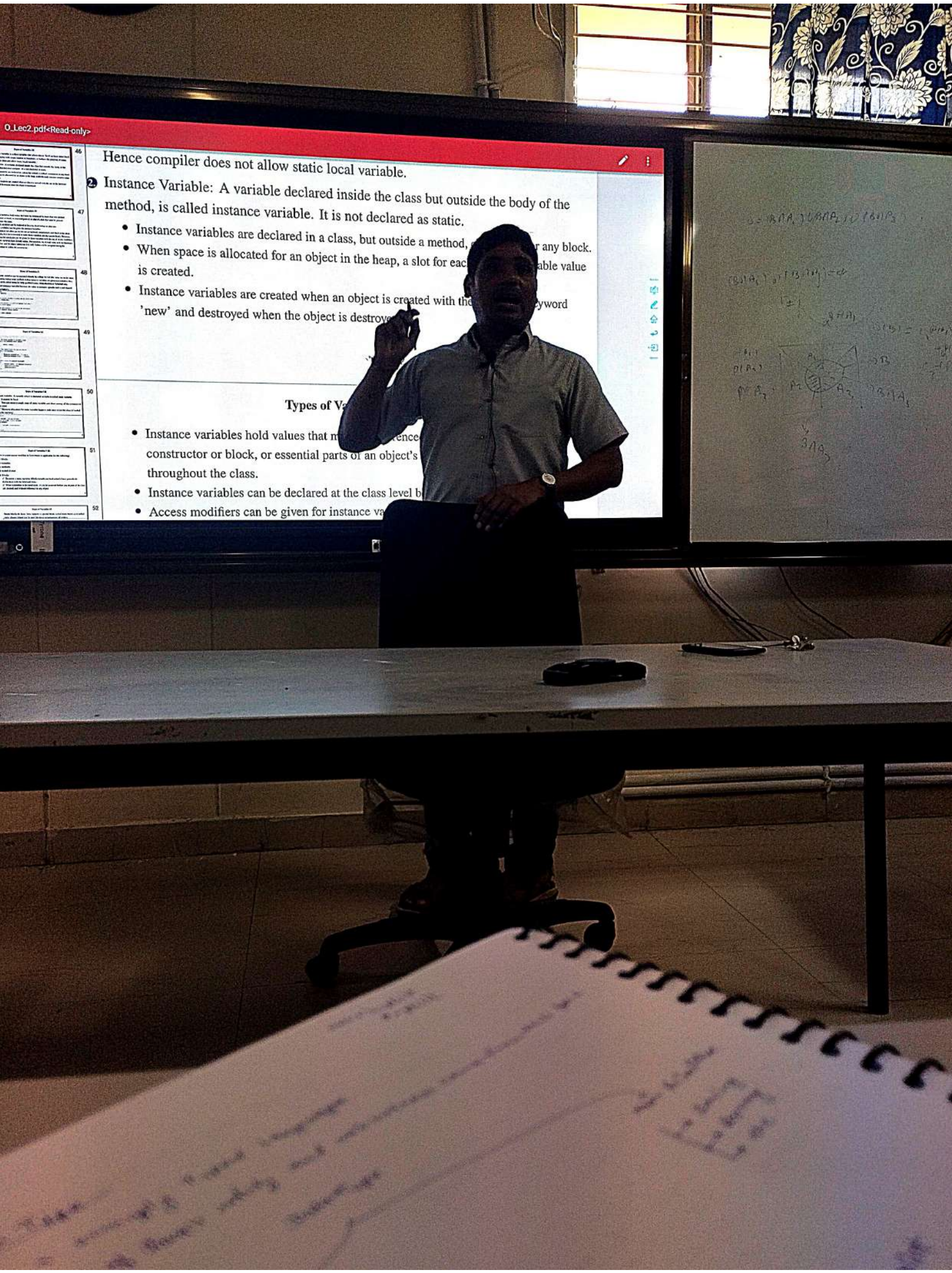

O_Lec2.pdf<Read-only>
38
39
40
41
42
43
44
45
46
Types of Variables II
1 Local Variable: A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
Can a local variable be defined with " static " keyword?
If no, then why ????????

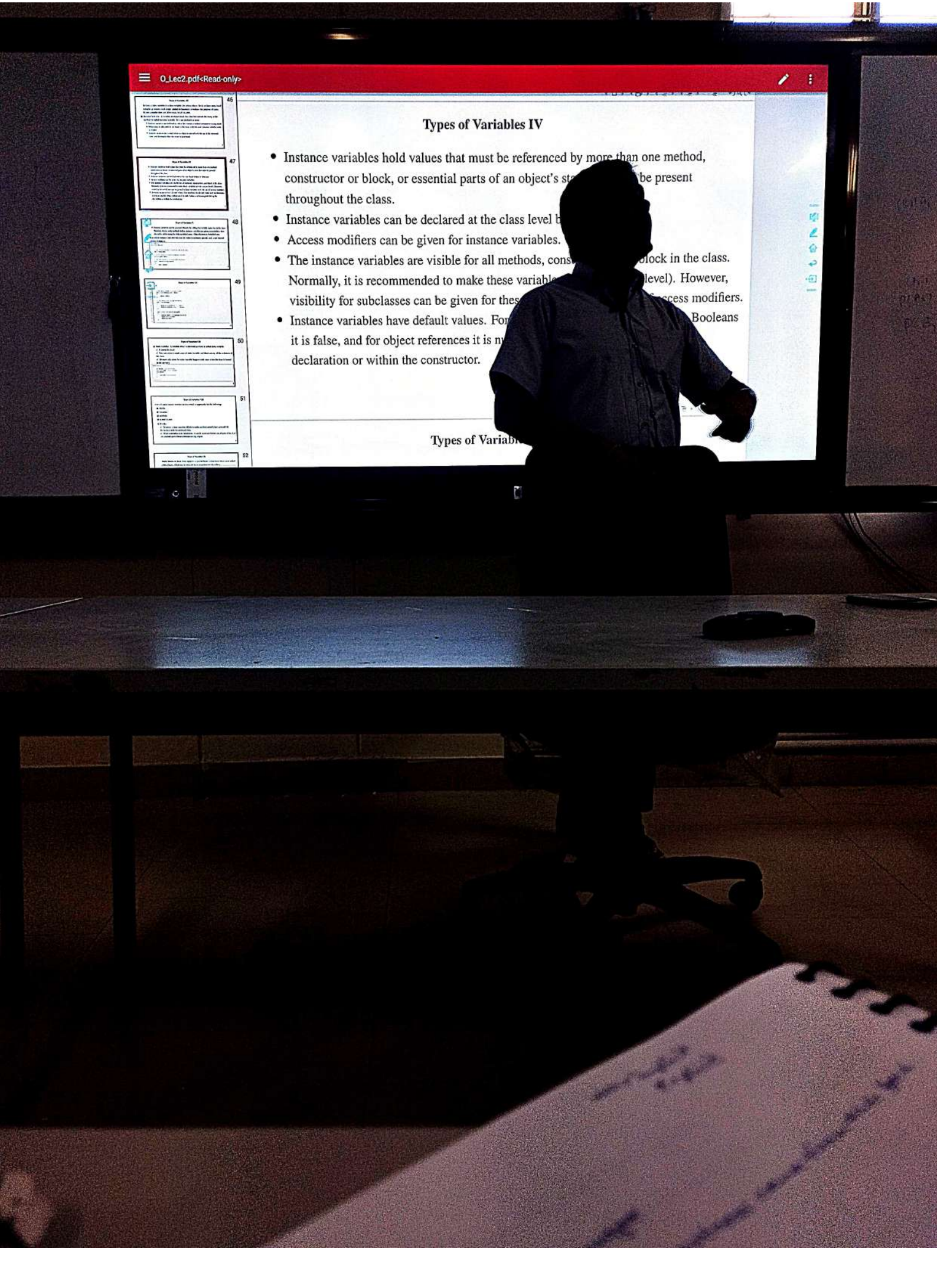

O_Lec2.pdf<Read-only>
Types of Variables II
Local Variable: A variable declared inside the body of the metho
variable. You can use this variable only within that method and t
the class aren't even aware that the variable exists.
Can a local variable be defined with " static " keyword?
If no, then why ????????
sum(int a, int b)
static int c;
c=a+b;
main( )
int a,b;
sum(5

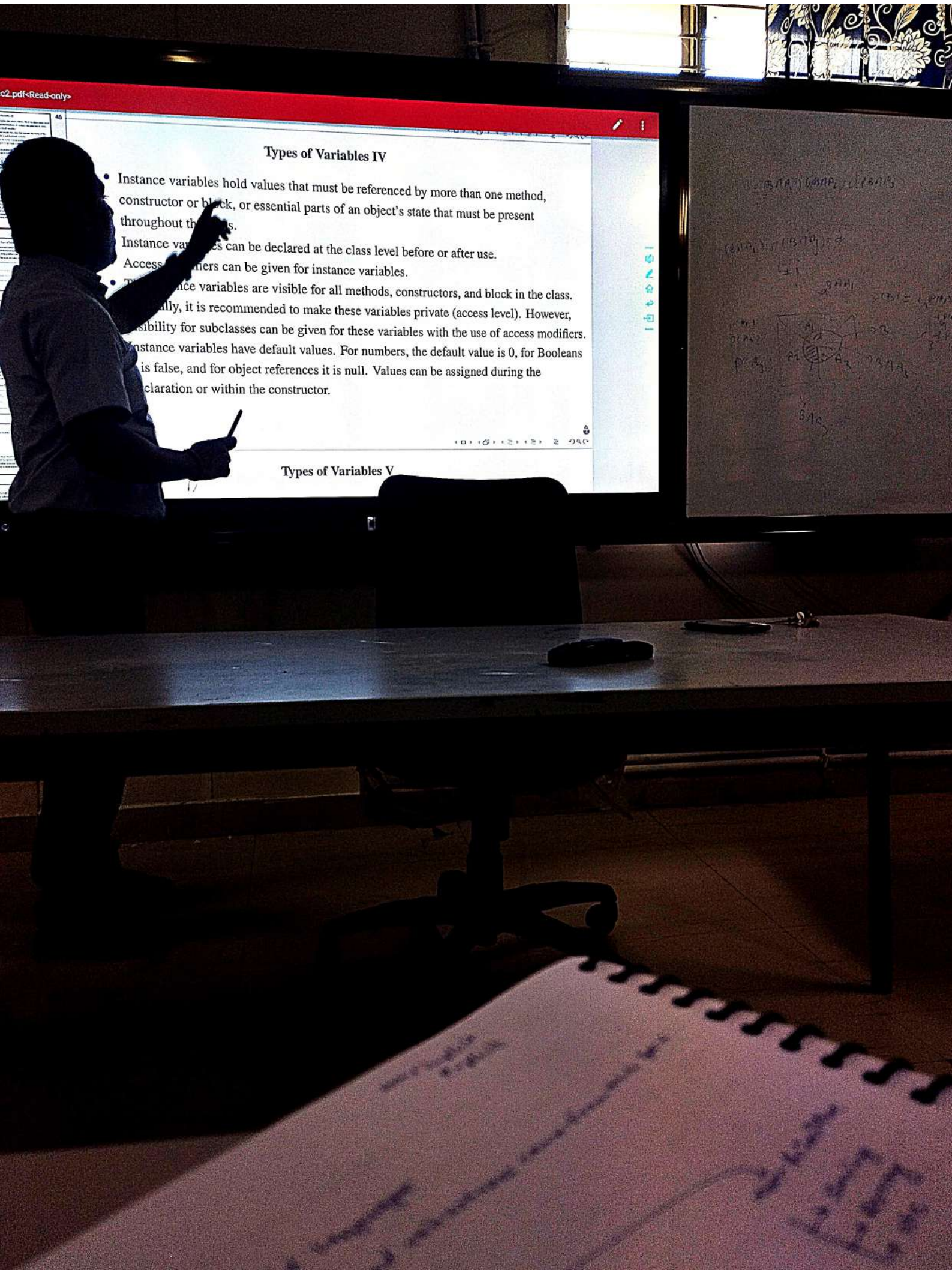

O_Lec2.pdf<Read-only>
In Java, a static variable is a class variable (for whole class). So if we have static local variable (a variable with scope limited to function), it violates the purpose of static. Hence compiler does not allow static local variable.
2. Instance Variable: A variable declared inside the class but outside
he method, is called instance variable. It is not declared as static.
Instance variables are declared in a class, but outside a method, co
block.
When space is allocated for an object in the heap, a slot for each inst
value is created.
Instance variables are created when an object is created with t
'new' and destroyed when the object is destroyed.
Types of Variables IV
Instance variables hold values that must be referenced by mo
constructor or block, or essential parts

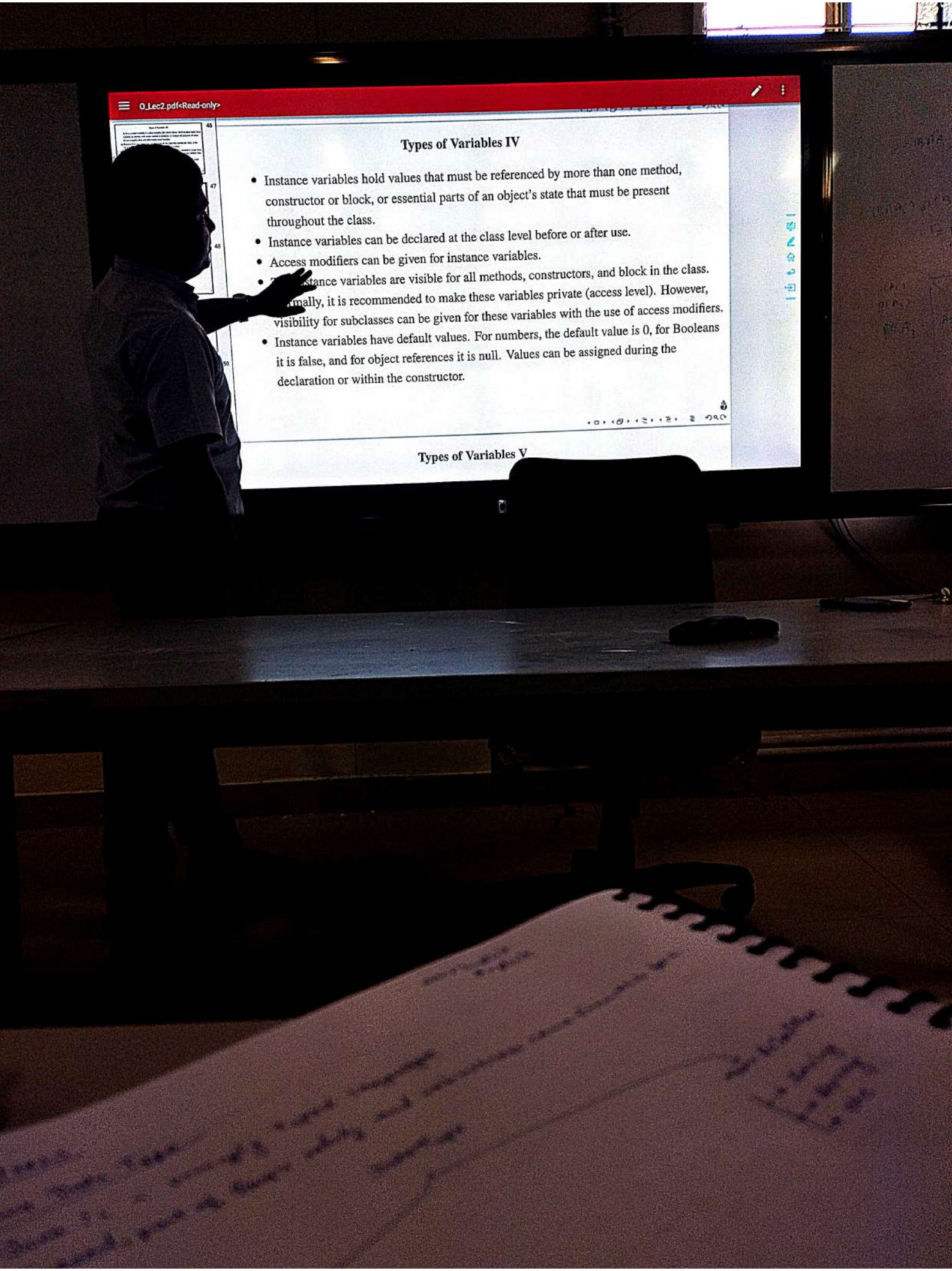

Q_Lec2.pdf<Read-only>
In Java, a static variable is a class variable (for whole class). So if we have static local variable (a variable with scope limited to function), it violates the purpose of static. Hence compiler does not allow static local
Instance Variable: A variable declared in
but outside the body of the method, is called instance variable. It is
s static.
Instance variables are declared in a class
method, constructor or any block.
When space is allocated for an object in t
t for each instance variable value is created.
Instance variables are created when
e use of the keyword 'new' and destroyed when the
Types o
Instance variables hold values that
e method, constructor or block, or essential p
ent

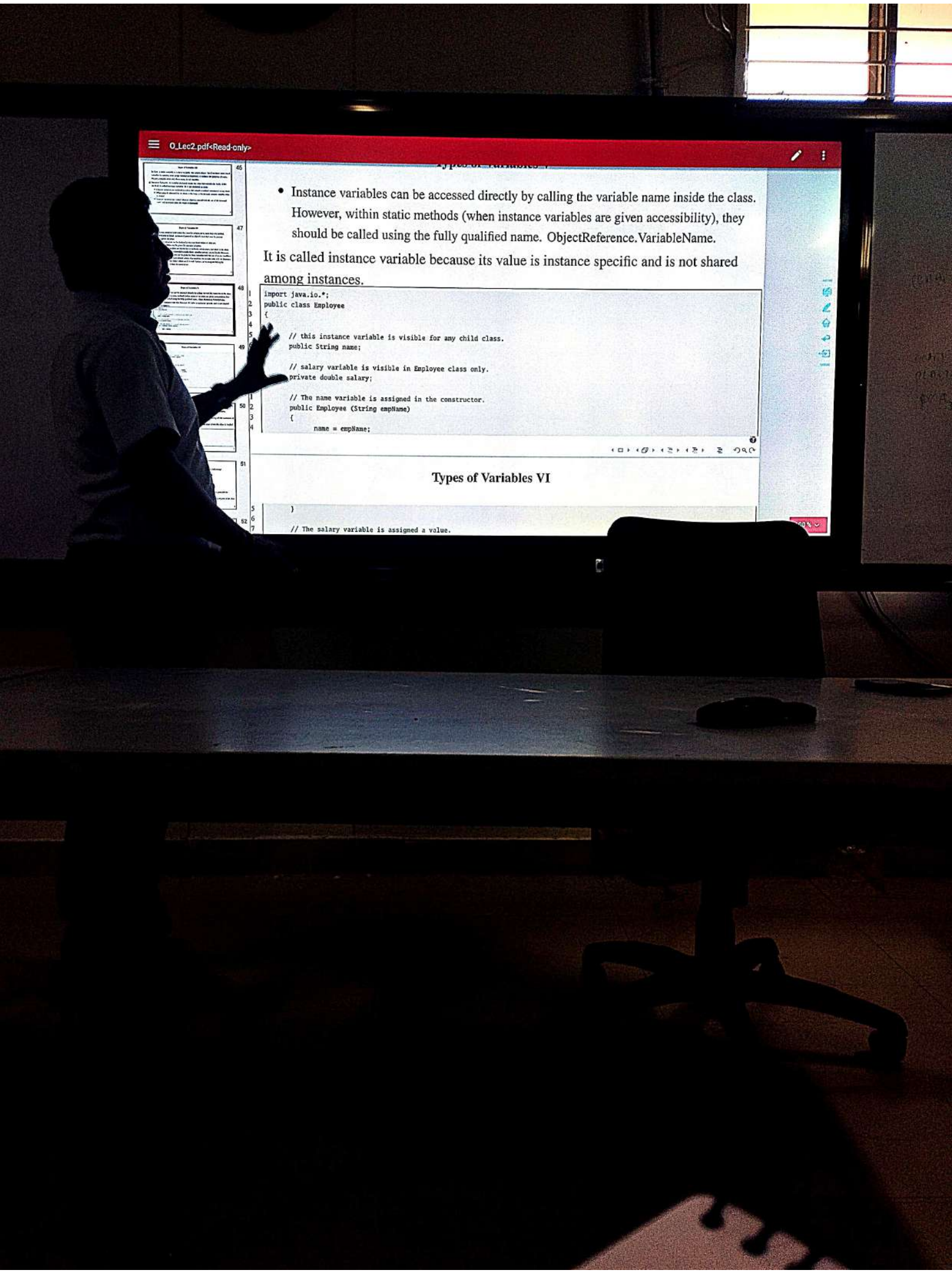

O_Lec2.pdf<Read-only>
variable (a variable with scope limited to function), it violates the purpose of static. Hence compiler does not allow static local variable.
2 Instance Variable: A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
Instance variables are declared in a class, but outside a method, constructor or any block.
When space is allocated for an object in the heap, a slot for each instance variable value is created.
Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
int *a;
*a=5;
Types of Variables IV
Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present


O_Lec2.pdf<Read-only>
Hence compiler does not allow static local variable.
2. Instance Variable: A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
• Instance variables are declared in a class, but outside a method,
r any block.
• When space is allocated for an object in the heap, a slot for eac
able value is created.
• Instance variables are created when an object is created with the
yword 'new' and destroyed when the object is destroy
Types of V
• Instance variables hold values that n
ence
constructor or block, or essential parts of an object's
throughout the class.
• Instance variables can be declared at the class level b
• Access modifiers can be given for instance va


O_Lec2.pdf<Read-only>
Types of Variables IV
• Instance variables hold values that must be referenced by more than one method,
constructor or block, or essential parts of an object's st be present
throughout the class.
• Instance variables can be declared at the class level b
• Access modifiers can be given for instance variables.
• The instance variables are visible for all methods, cons lock in the class.
Normally, it is recommended to make these variabl level). However,
visibility for subclasses can be given for thes cess modifiers.
• Instance variables have default values. For Booleans
it is false, and for object references it is n
declaration or within the constructor.
Types of Variabl


c2.pdf<Read-only>
Types of Variables IV
Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
Instance variables can be declared at the class level before or after use.
Access modifiers can be given for instance variables.
Instance variables are visible for all methods, constructors, and block in the class. Usually, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
Types of Variables V


0_Lec2.pdf<Read-only>
Types of Variables IV
Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
Instance variables can be declared at the class level before or after use.
Access modifiers can be given for instance variables.
Instance variables are visible for all methods, constructors, and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
Types of Variables V


O_Lec2.pdf<Read-only>
Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.
It is called instance variable because its value is instance specific and is not shared among instances.
import java.io.*;
public class Employee
{
// this instance variable is visible for any child class.
public String name;
// salary variable is visible in Employee class only.
private double salary;
// The name variable is assigned in the constructor.
public Employee (String empName)
{
name = empName;
Types of Variables VI
}
// The salary variable is assigned a value.

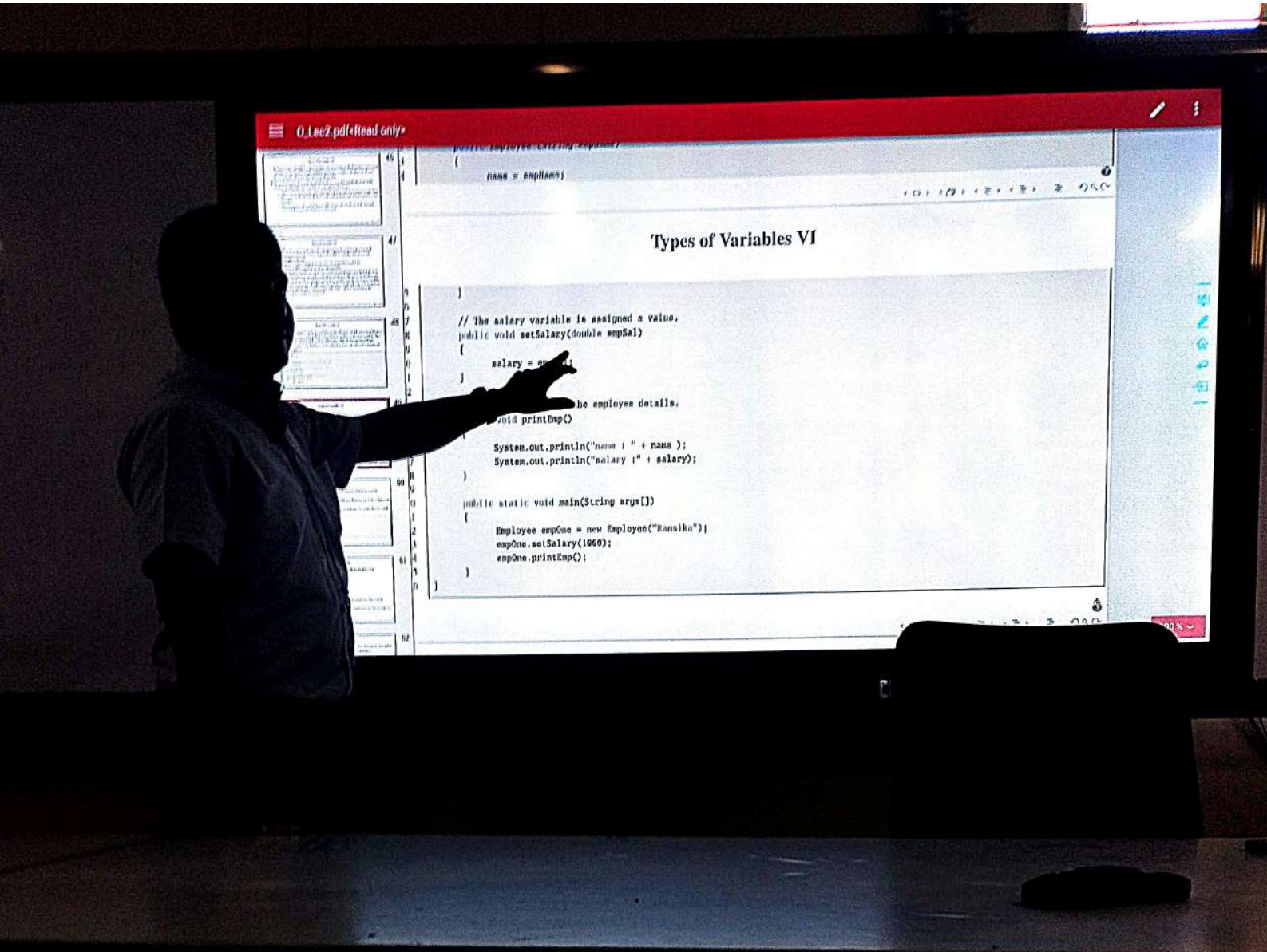
O_Lec2.pdf<Read-only>

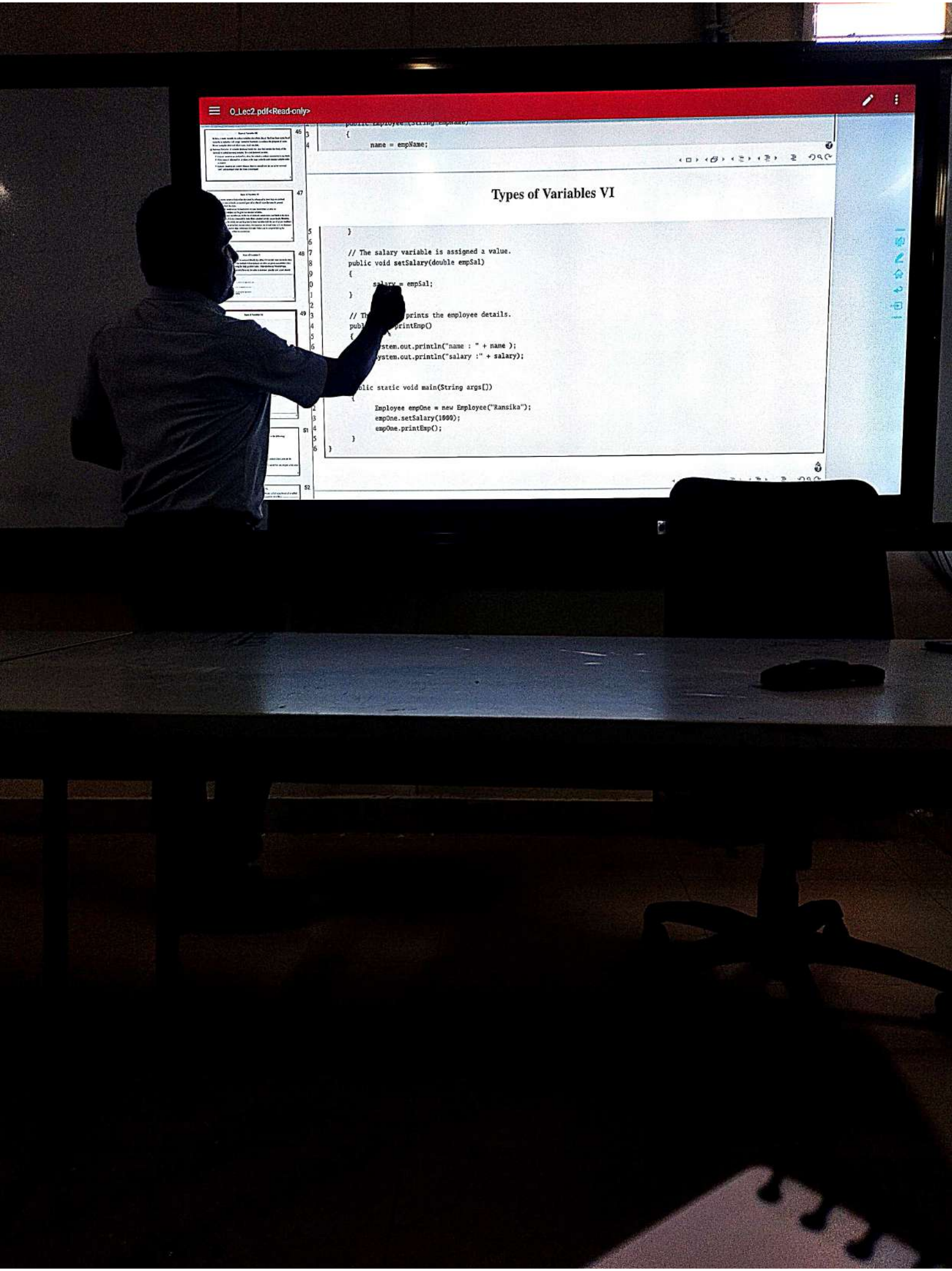
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.

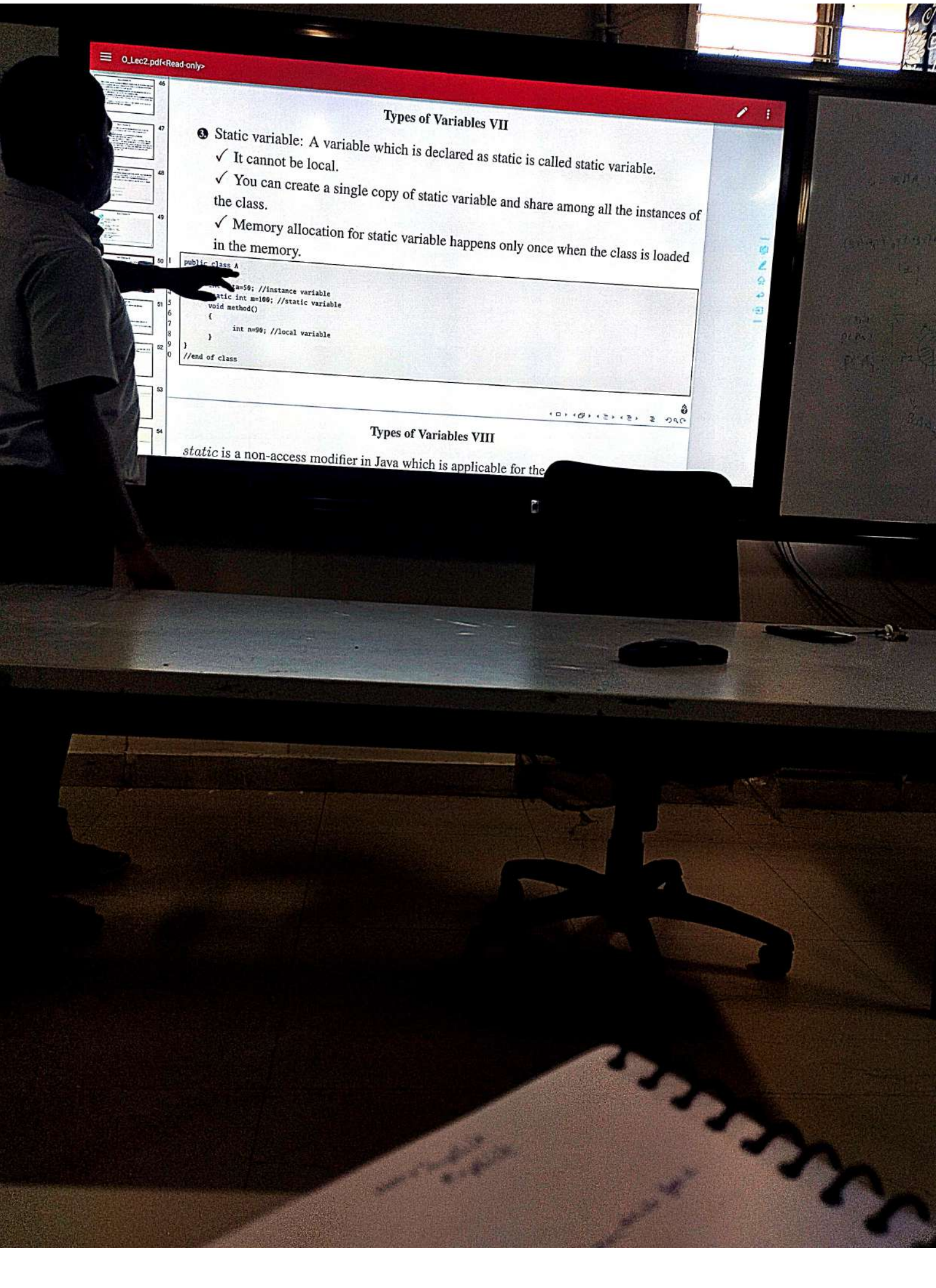
It is called instance variable because its value is instance specific and is not shared among instances.

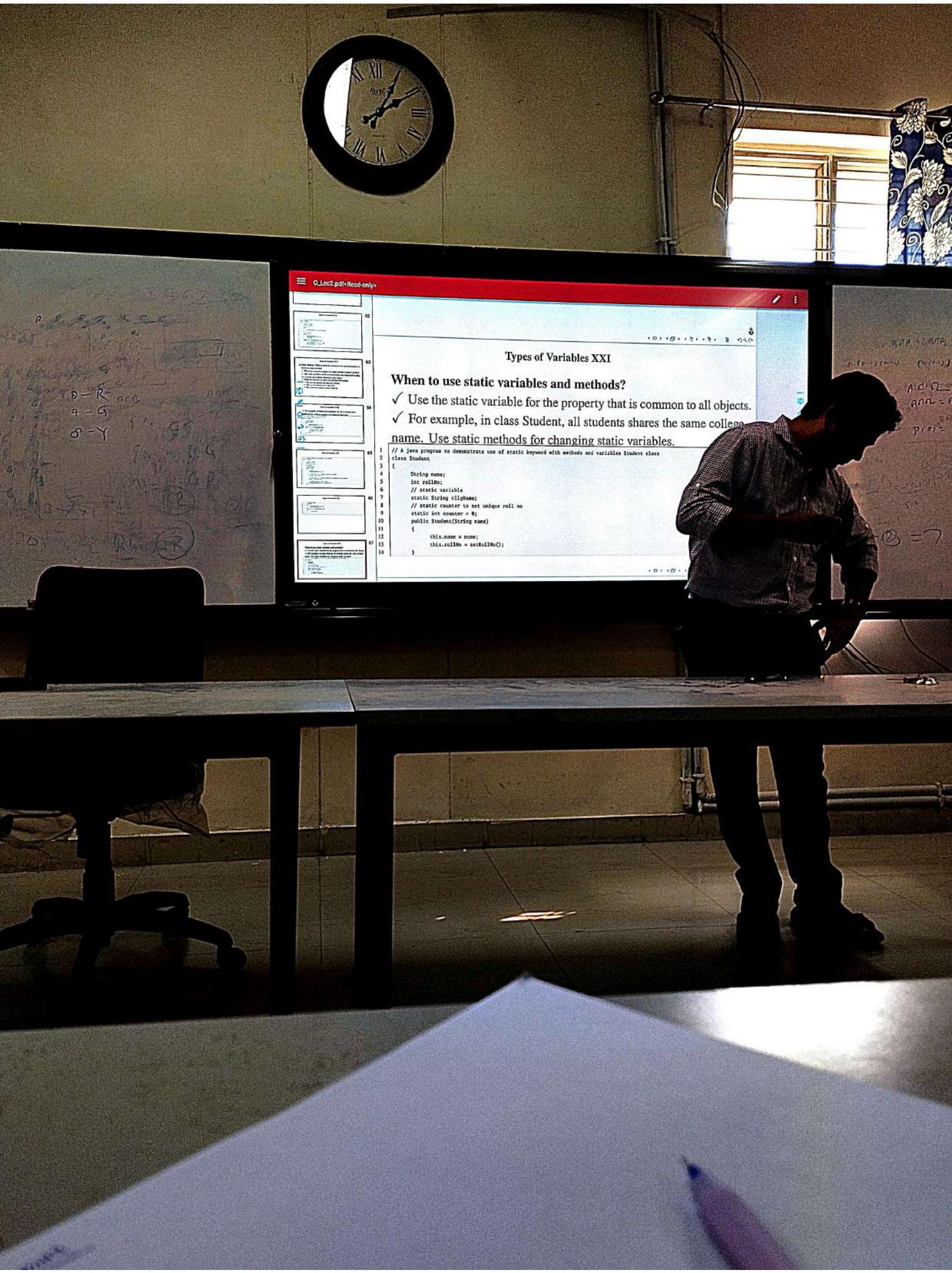
```
import java.io.*;
public class Employee
{

      // this instance variable is visible for any child class.
      public String name;

      // salary variable is visible in Employee class only.
      private double salary;

      // The name variable is assigned in the constructor.
      public Employee (String empName)
      {
            name = empName;
      }

      // The salary variable is assigned a value.
```

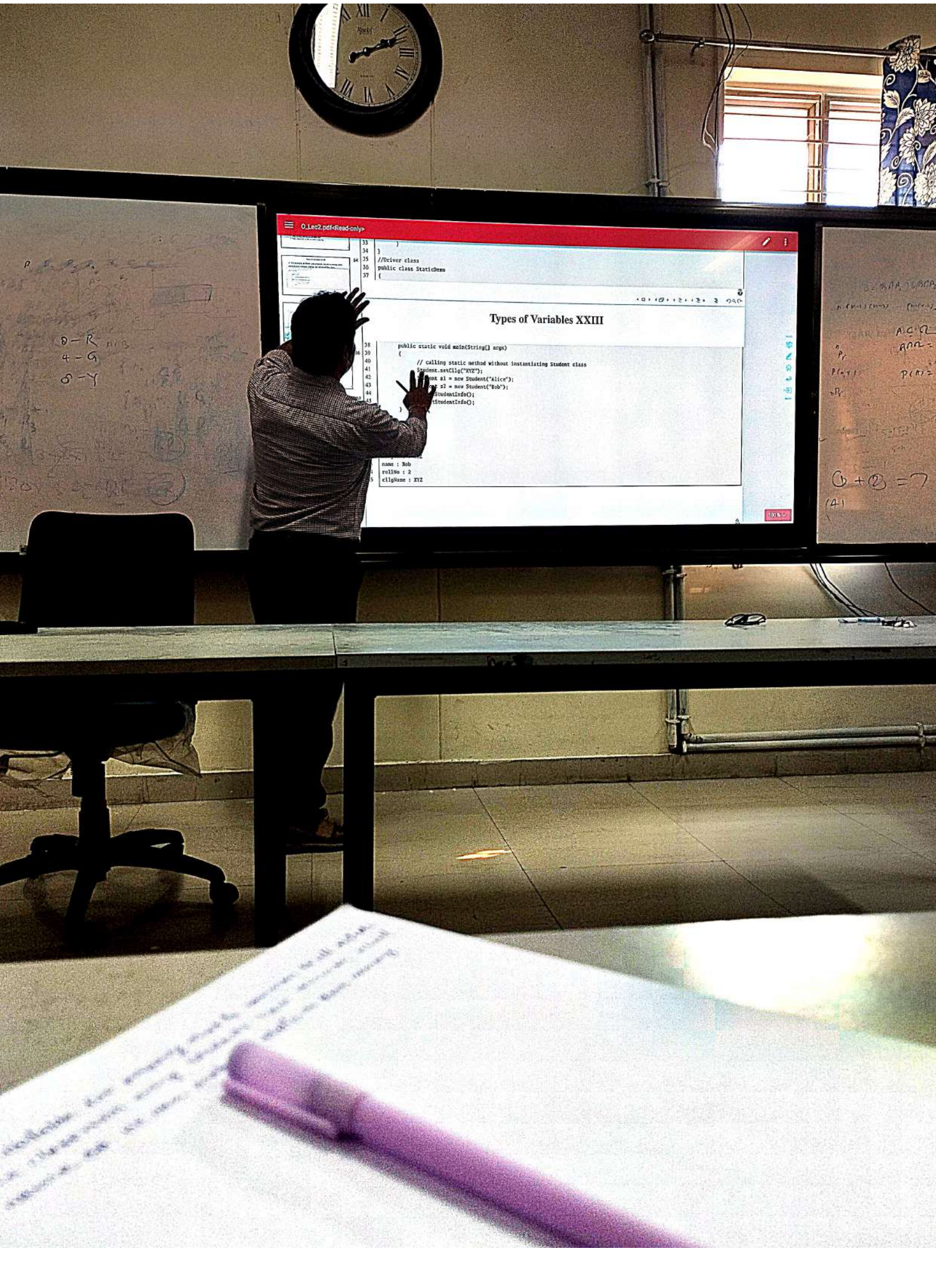
Types of Variables VI

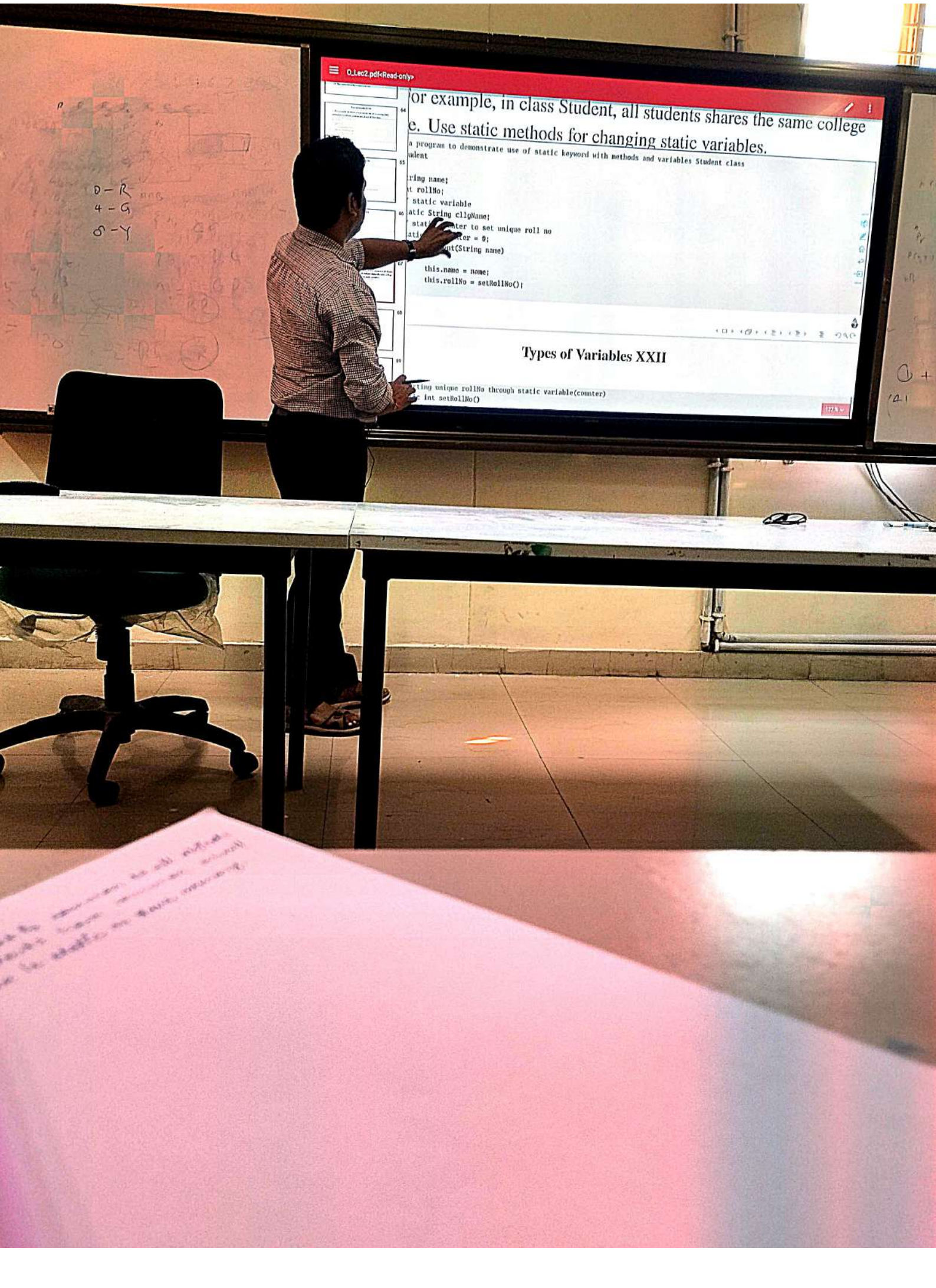

0_Lec2.pdf•Read only•
Types of Variables VI
// The salary variable is assigned a value.
public void setSalary(double empSal)
{
salary = empSal;
}
/* This method prints the employee details.
*/
public void printEmp()
{
System.out.println("name : " + name);
System.out.println("salary :" + salary);
}
public static void main(String args[])
{
Employee empOne = new Employee("Ransika");
empOne.setSalary(1000);
empOne.printEmp();
}
}

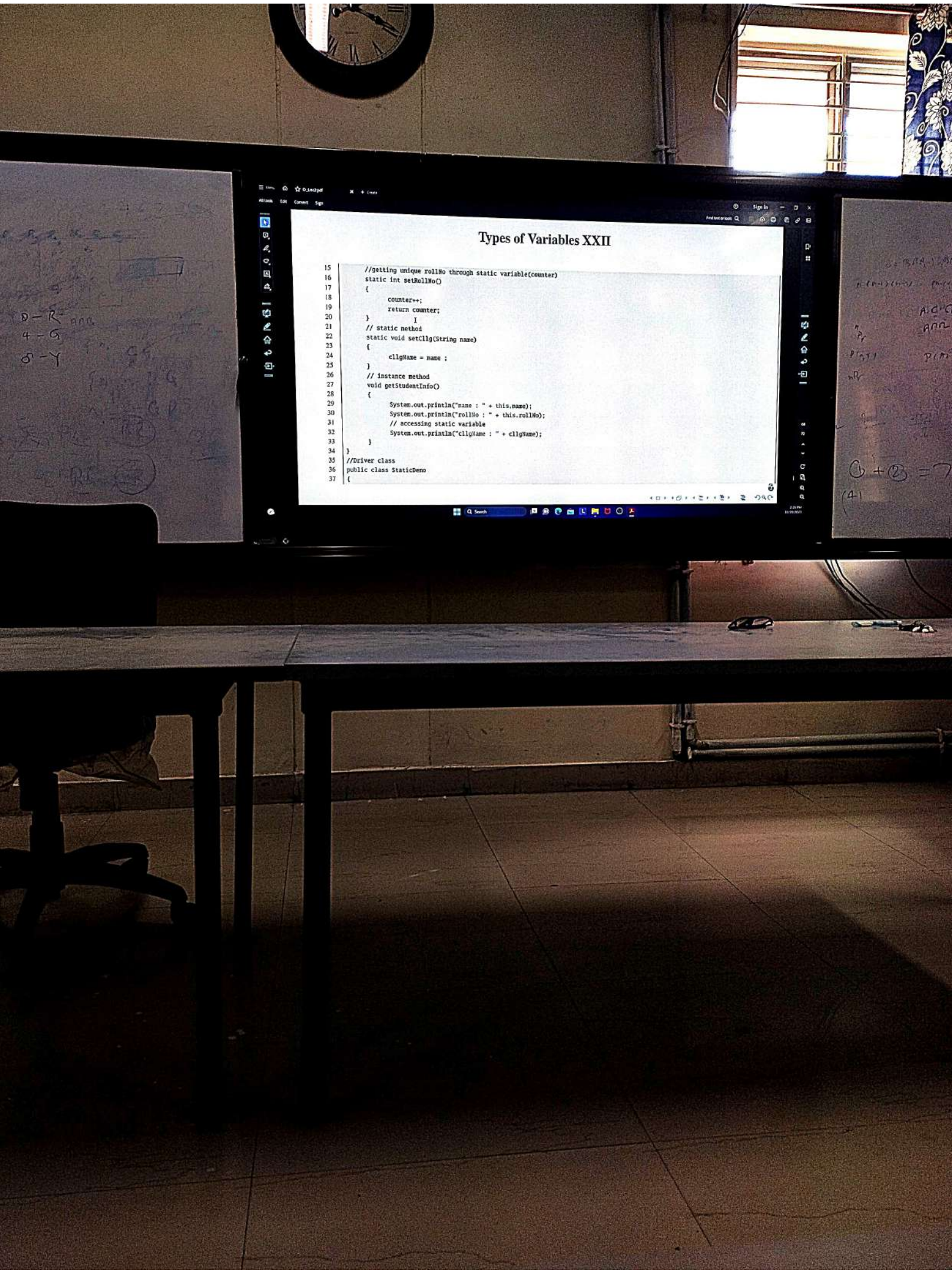

0_Lec2.pdf<Read-only>
Types of Variables VI
name = empName;
}
// The salary variable is assigned a value.
public void setSalary(double empSal)
{
salary = empSal;
}
prints the employee details.
printEmp()
ystem.out.println("name : " + name);
ystem.out.println("salary :" + salary);
blic static void main(String args[])
Employee empOne = new Employee("Ransika");
empOne.setSalary(1000);
empOne.printEmp();
}

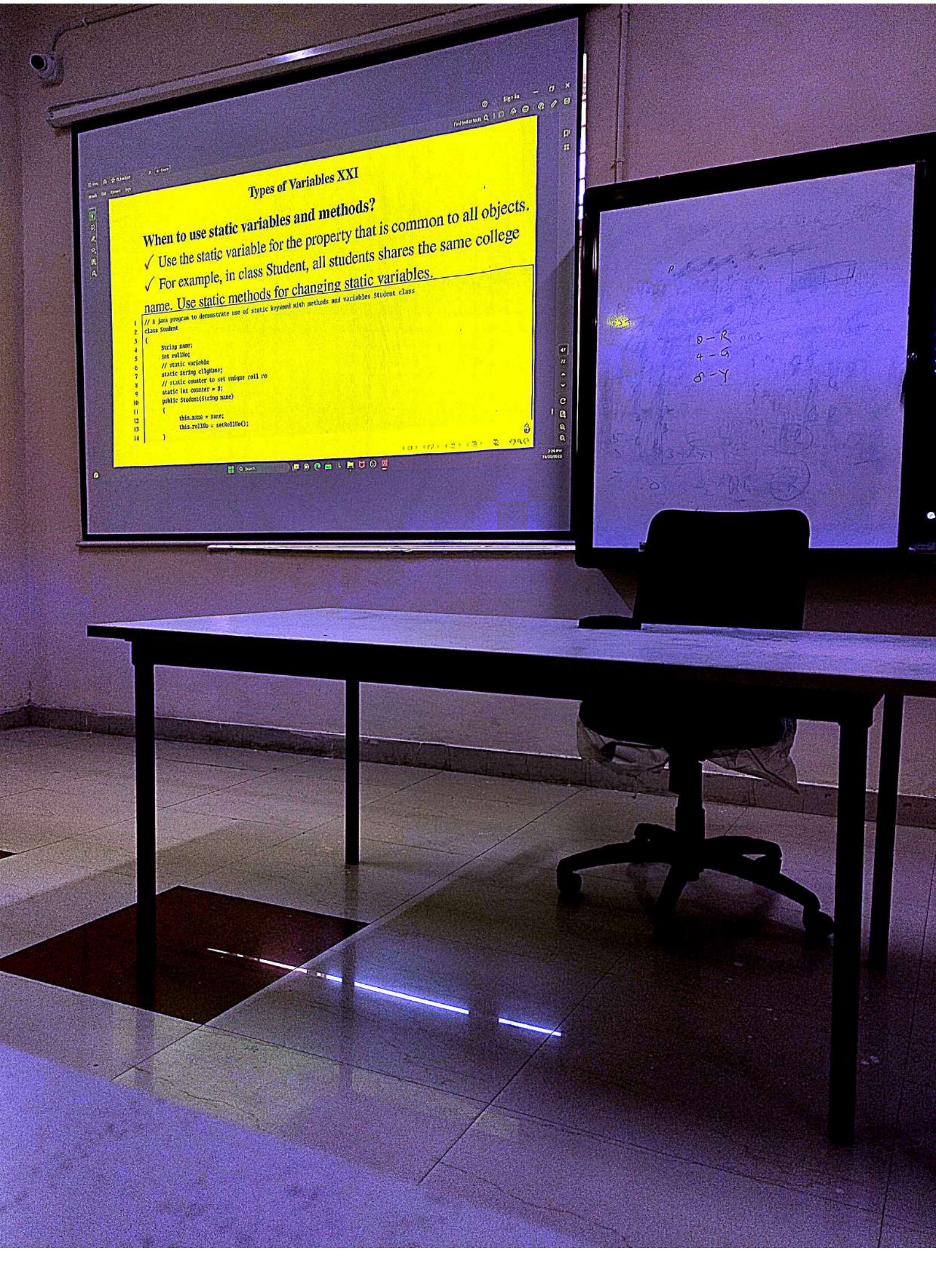

0_Lec2.pdf<Read-only>
Types of Variables VII
3. Static variable: A variable which is declared as static is called static variable.
✓ It cannot be local.
✓ You can create a single copy of static variable and share among all the instances of the class.
✓ Memory allocation for static variable happens only once when the class is loaded in the memory.
public class A
a=50; //instance variable
atic int m=100; //static variable
void method()
{
int n=90; //local variable
}
}
//end of class
Types of Variables VIII
static is a non-access modifier in Java which is applicable for the

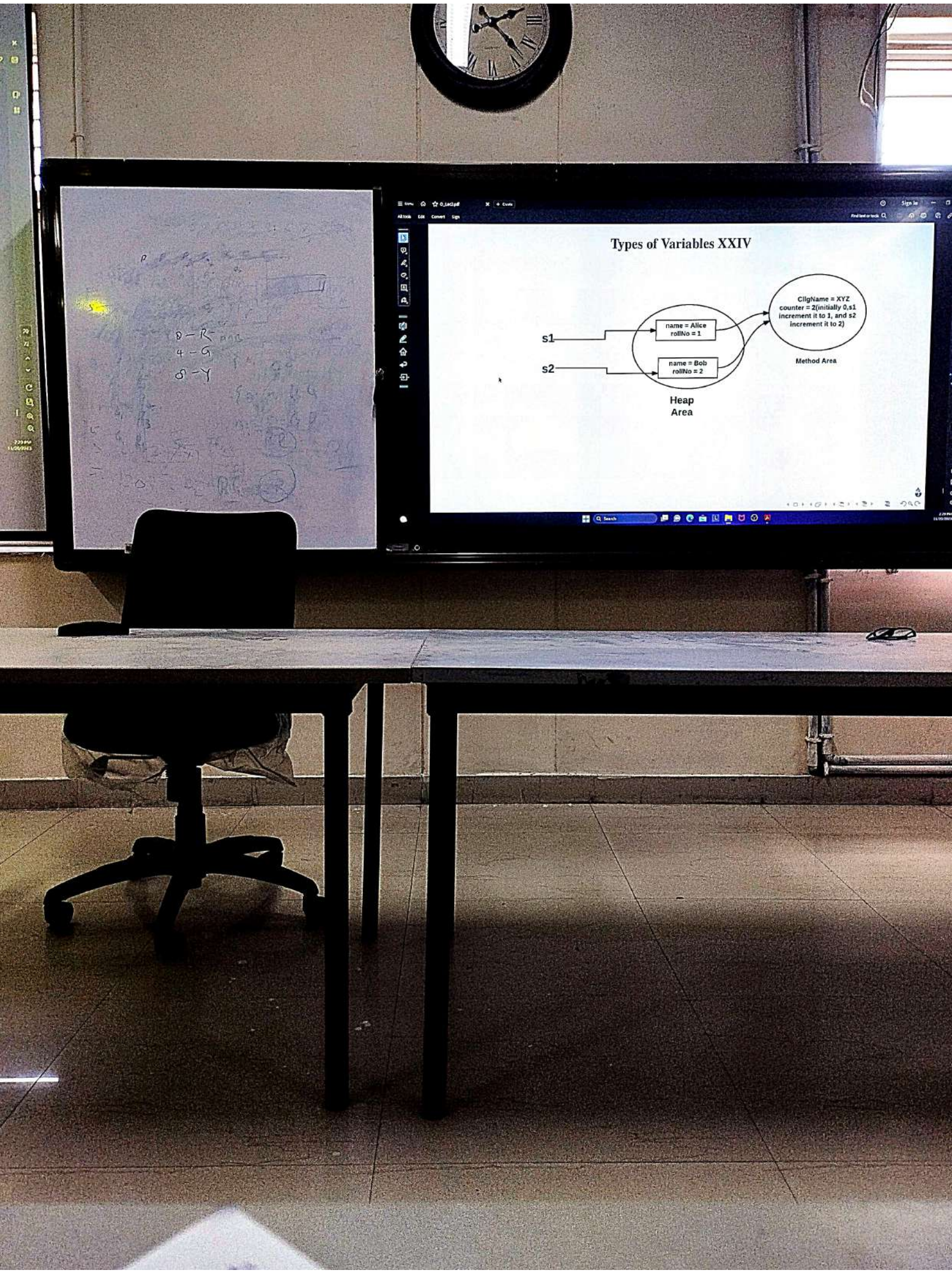

O_Lec2.pdf<Read-only>
Types of Variables XXI
When to use static variables and methods?
✓ Use the static variable for the property that is common to all objects.
✓ For example, in class Student, all students shares the same college
name. Use static methods for changing static variables.
// A java program to demonstrate use of static keyword with methods and variables Student class
class Student
{
String name;
int rollNo;
// static variable
static String cllgName;
// static counter to set unique roll no
static int counter = 0;
public Student(String name)
{
this.name = name;
this.rollNo = setRollNo();
}


0_Lec2.pdf<Read-only>
Types of Variables XXIII
public static void main(String[] args)
{
// calling static method without instantiating Student class
Student.setClg("XYZ");
name : Bob
rollNo : 2
clgName : XYZ

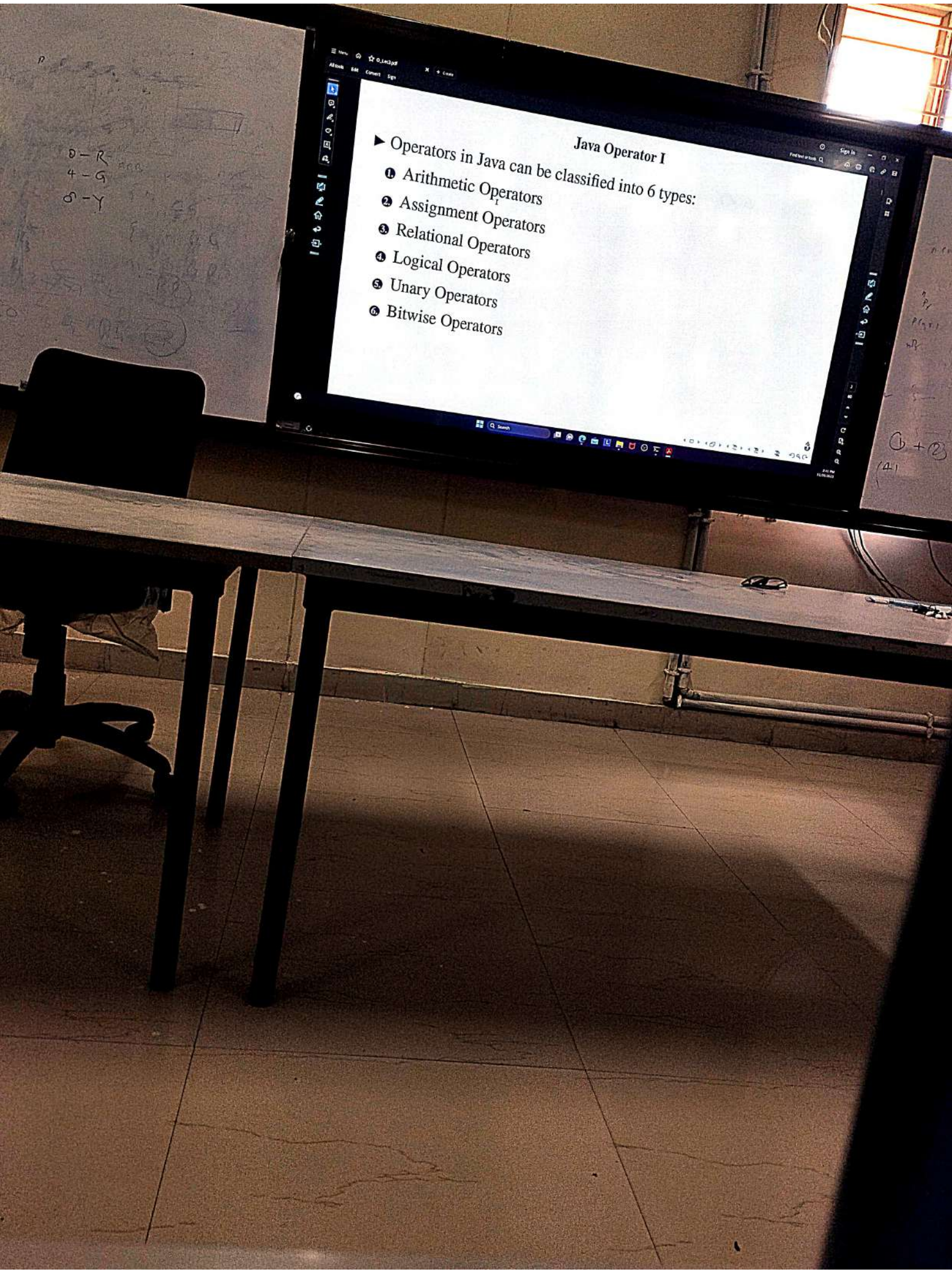

O_Lec2.pdf<Read-only>
or example, in class Student, all students shares the same college
e. Use static methods for changing static variables.
a program to demonstrate use of static keyword with methods and variables Student class
udent
ring name;
t rollNo;
static variable
atic String cllgName;
ter to set unique roll no
ter = 0;
nt(String name)
this.name = name;
this.rollNo = setRollNo();
Types of Variables XXII
ting unique rollNo through static variable(counter)
int setRollNo()


Types of Variables XXII
//getting unique rollNo through static variable(counter)
static int setRollNo()
{
counter++;
return counter;
}
// static method
static void setCllg(String name)
{
cllgName = name ;
}
// instance method
void getStudentInfo()
{
System.out.println("name : " + this.name);
System.out.println("rollNo : " + this.rollNo);
// accessing static variable
System.out.println("cllgName : " + cllgName);
}
}
//Driver class
public class StaticDemo
{


Types of Variables XXI
When to use static variables and methods?
✓ Use the static variable for the property that is common to all objects.
✓ For example, in class Student, all students shares the same college
name. Use static methods for changing static variables.
// A Java program to demonstrate use of static keyword with methods and variables Student class
class Student
{
String name;
int rollNo;
// static variable
static String clgName;
// static counter to set unique roll no
static int counter = 0;
public Student(String name)
{
this.name = name;
this.rollNo = setRollNo();
}


Types of Variables XXIV
s1
s2
name = Alice
rollNo = 1
name = Bob
rollNo = 2
Heap
Area
ClgName = XYZ
counter = 2(initially 0,s1
increment it to 1, and s2
increment it to 2)
Method Area


Types of Variables XXX
9. Static nested classes : We can not declare top-level class with a static modifier, but can declare nested classes as static. Such type of classes are called Nested static classes.


Java Operator I
Operators in Java can be classified into 6 types:
1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Unary Operators
6. Bitwise Operators


Java Operator II
Java Arithmetic Operators: Arithmetic operators are used to perform arithmetic operations on variables and data.
Operator | Operation
+ | Addition
- | Subtraction
* | Multiplication
/ | Division
% | Modulo Operation (Remainder after division)

Java Operator VI

Java Assignment Operators: Assignment operators are
assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Java Operator VI

Java Assignment Operators: Assignment operators are us
assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Java Operator IX

Java Relational Operators: Relational operator
relationship between two operands.

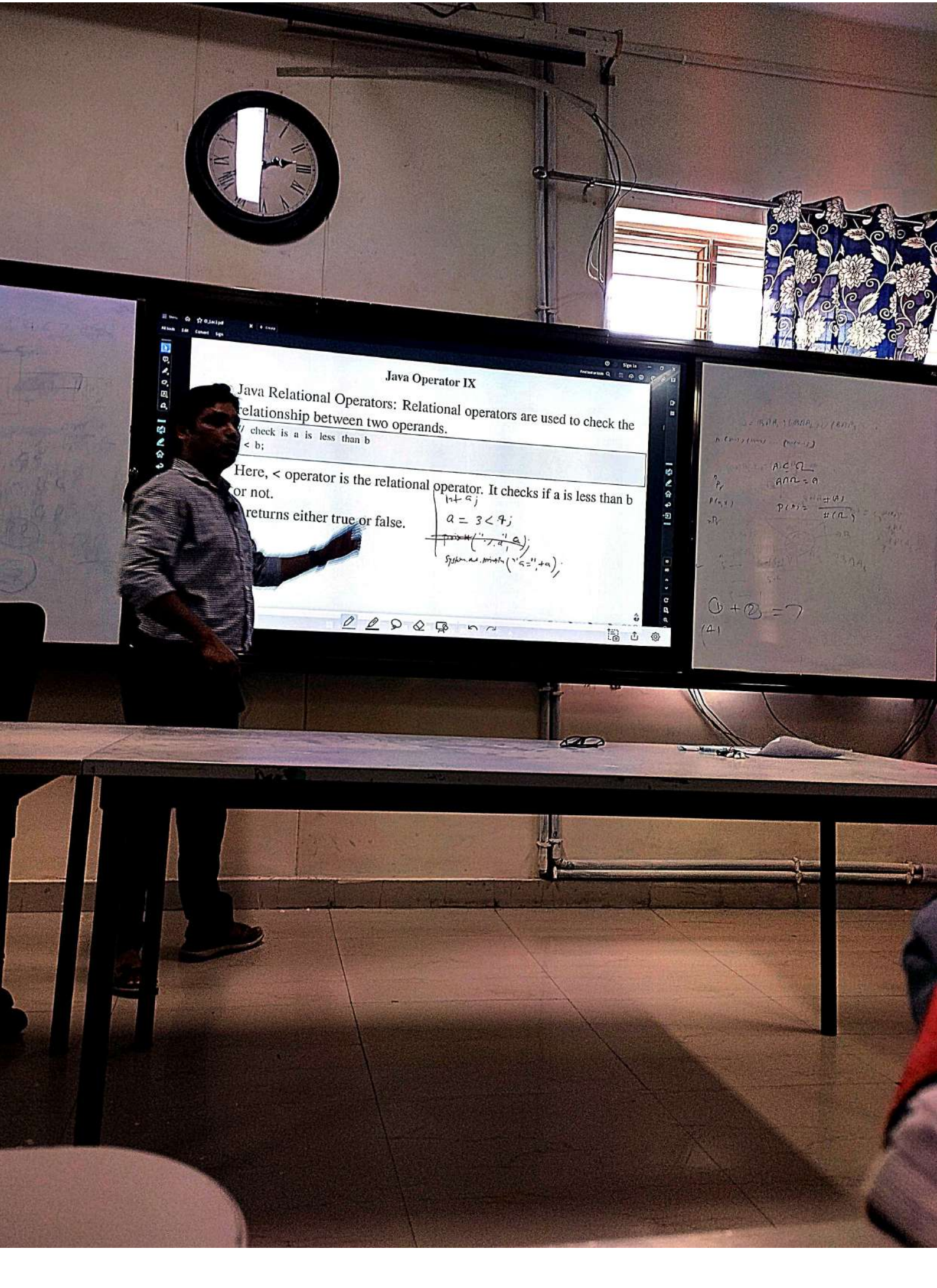
It checks if a is less than b

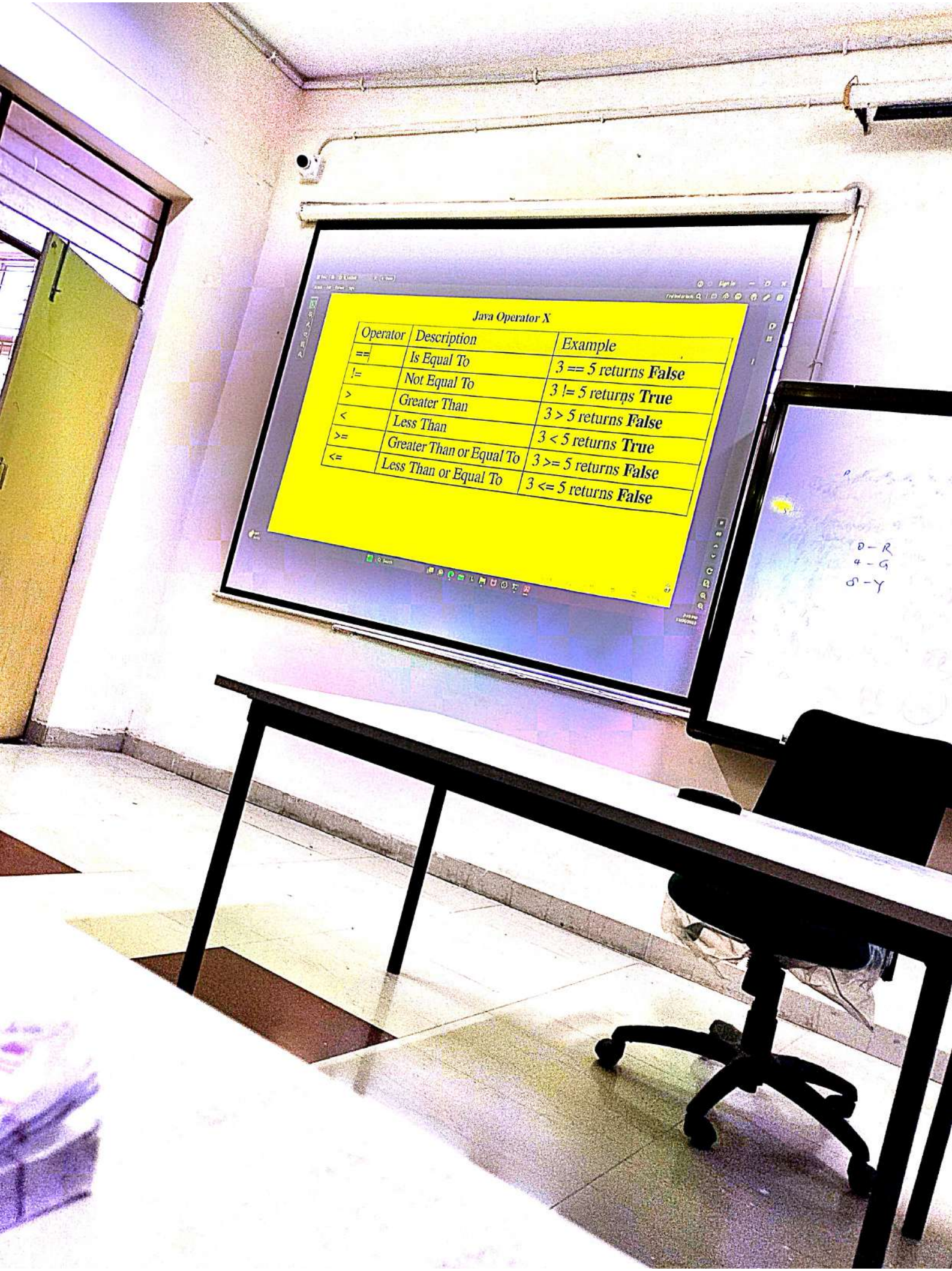
a < b;

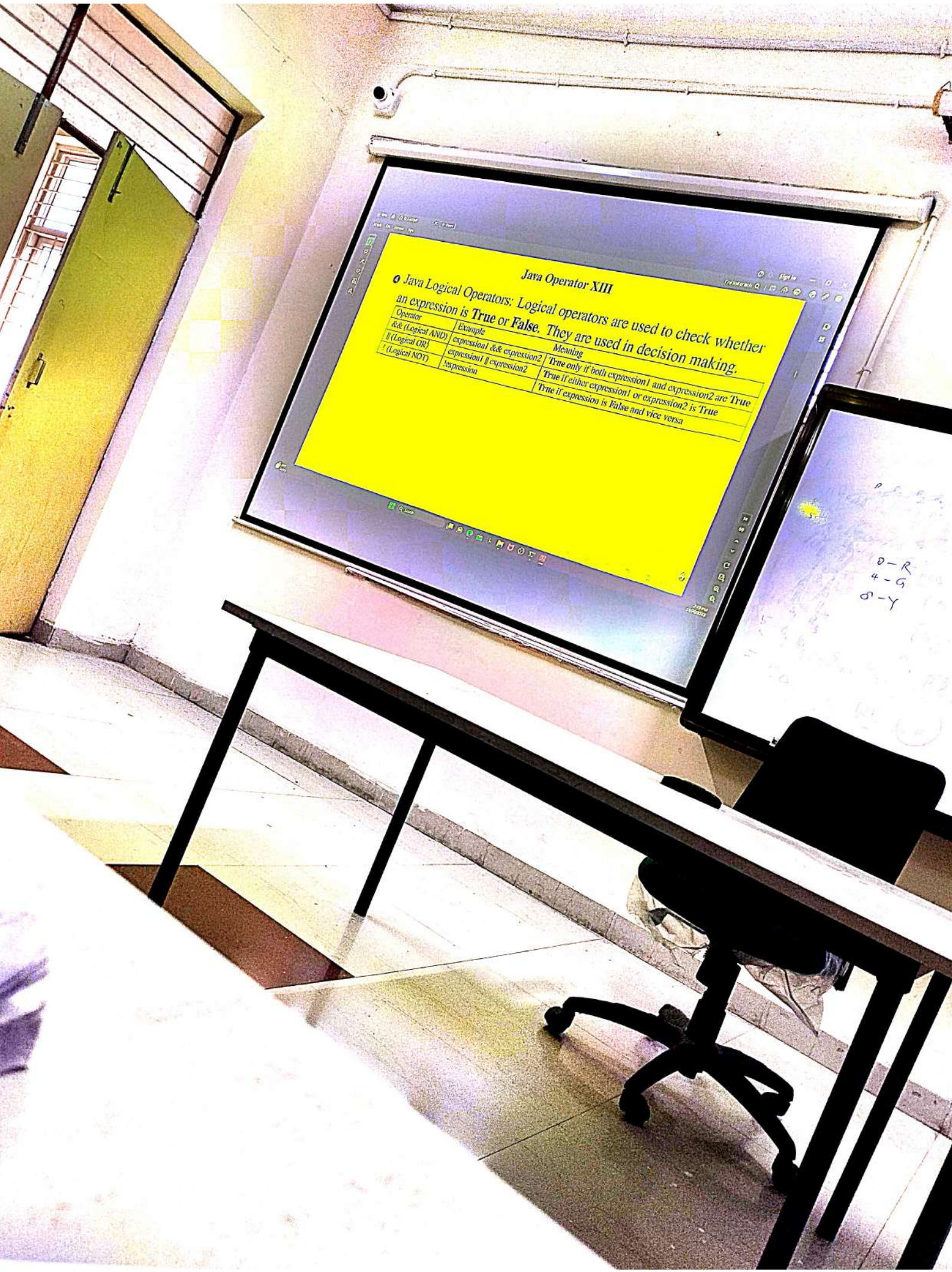
Here, < operator is the relational operator. It checks if a is
or not.

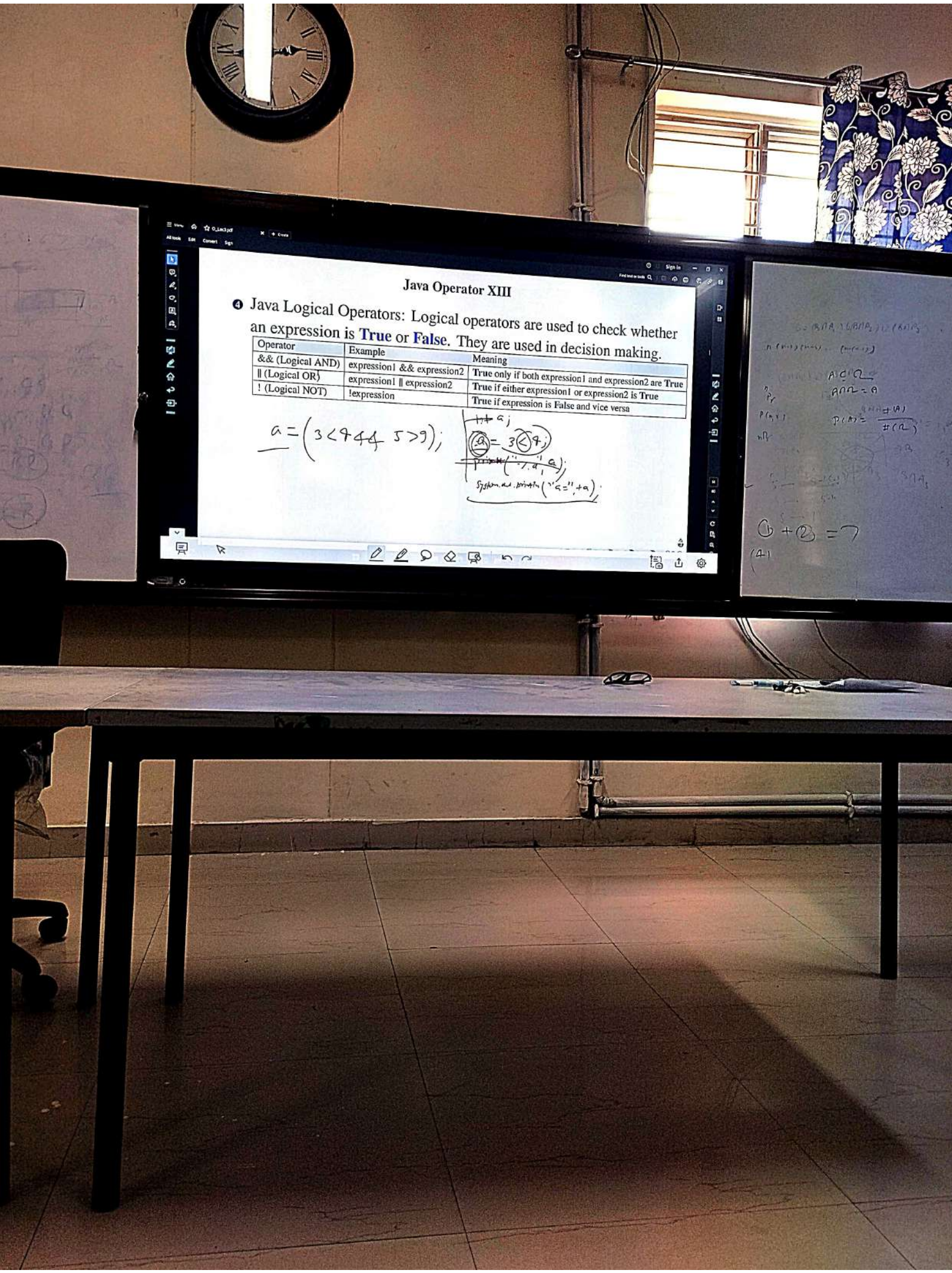
It returns either true or false.

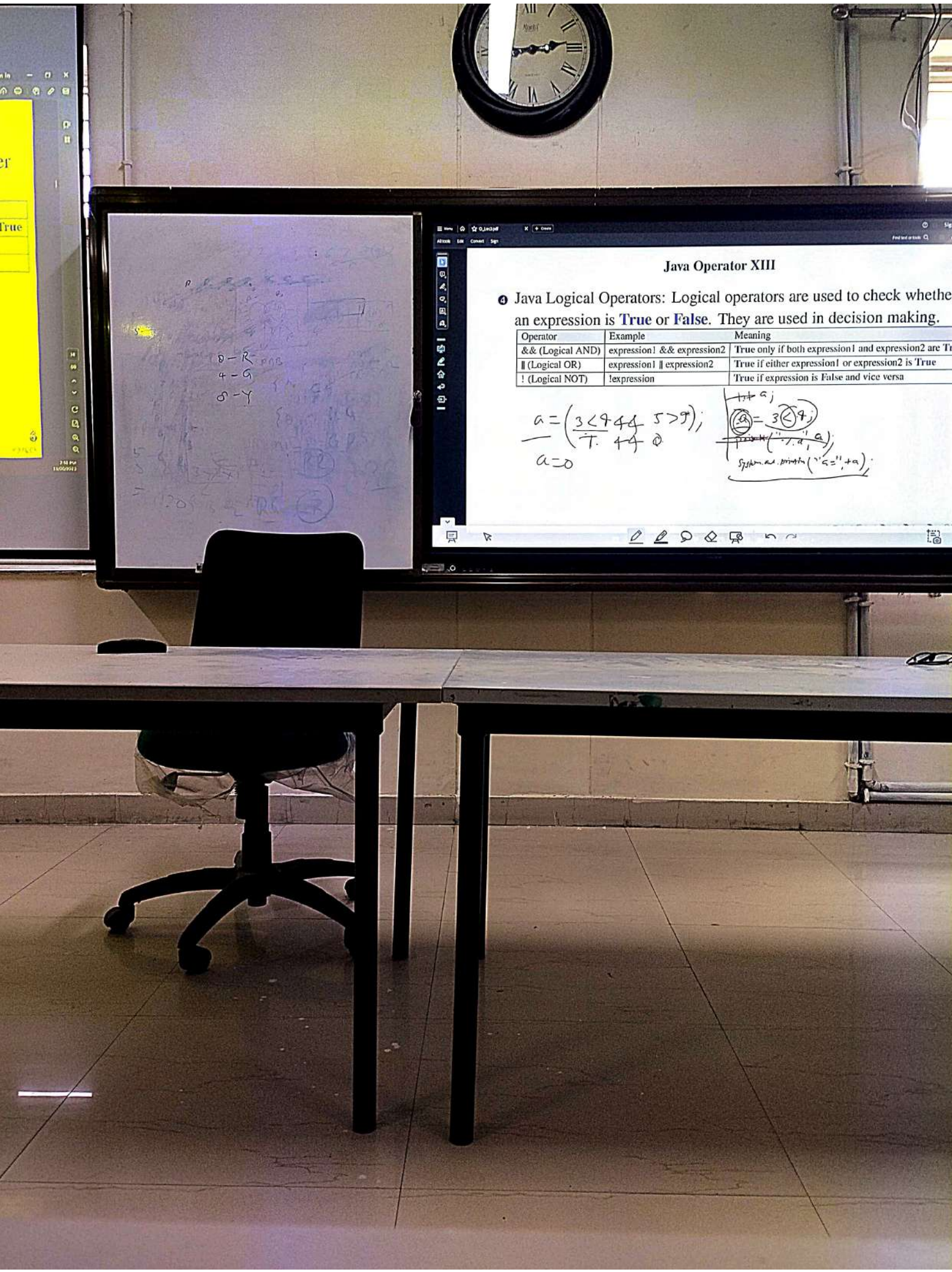

Java Operator IX
Java Relational Operators: Relational operators are used to check the relationship between two operands.
// check is a is less than b
a < b;
Here, < operator is the relational operator. It checks if a is less than b or not.
It returns either true or false.


Java Operator IX
Java Relational Operators: Relational operators are used to check the relationship between two operands.
// check is a is less than b
< b;
Here, < operator is the relational operator. It checks if a is less than b or not.
returns either true or false.
int a;
a = 3 < 4;


Java Operator X
Operator | Description | Example
== | Is Equal To | 3 == 5 returns False
!= | Not Equal To | 3 != 5 returns True
> | Greater Than | 3 > 5 returns False
< | Less Than | 3 < 5 returns True
>= | Greater Than or Equal To | 3 >= 5 returns False
<= | Less Than or Equal To | 3 <= 5 returns False


Java Operator XIII
Java Logical Operators: Logical operators are used to check whether an expression is True or False. They are used in decision making.
Operator | Example | Meaning
&& (Logical AND) | expression1 && expression2 | True only if both expression1 and expression2 are True
|| (Logical OR) | expression1 || expression2 | True if either expression1 or expression2 is True
! (Logical NOT) | !expression | True if expression is False and vice versa
0 - R
4 - G
8 - Y


Java Operator XIII
Java Logical Operators: Logical operators are used to check whether an expression is True or False. They are used in decision making.
Operator | Example | Meaning
&& (Logical AND) | expression1 && expression2 | True only if both expression1 and expression2 are True
|| (Logical OR) | expression1 || expression2 | True if either expression1 or expression2 is True
! (Logical NOT) | !expression | True if expression is False and vice versa


Java Operator XIII
Java Logical Operators: Logical operators are used to check whethe
an expression is True or False. They are used in decision making.
Operator | Example | Meaning
&& (Logical AND) | expression1 && expression2 | True only if both expression1 and expression2 are Tr
|| (Logical OR) | expression1 || expression2 | True if either expression1 or expression2 is True
! (Logical NOT) | !expression | True if expression is False and vice versa

Java Operator XVI

- Java Unary Operators: Unary operators are used with only one operand. For example, ++ is a unary operator that increases the value of a variable by 1. That is, ++5 will return 6.

Operator	Meaning
+	Unary plus: not necessary to use since numbers are positive without using it
-	Unary minus: inverts the sign of an expression
++	Increment operator: increments value by 1
--	Decrement operator: decrements value by 1
!	Logical complement operator: inverts the value of a boolean


Java Operator NIX
public class Operator
{
public static void main(String[] args)
{
int var1 = 5, var2 = 5;
// var1 is displayed
// Then, var1 is increased to 6.
System.out.println (var1++);
// var2 is increased to 6, then, var2 is displayed
System.out.println (++var2);
}
}
Output: ?

Java Operator XX
Java Bitwise Operators: Bitwise operators in Java are used to perform operations on individual bits.
Bitwise complement Operation of 85
~ 85 = 00100011 (In Binary)
~ 00100011
11011100 = 220 (In decimal)
Here, ~ is a bitwise operator. It inverts the value of each bit (0 to 1 and 1 to 0).

Java Operator XXI
Operator Description
Bitwise Complement
Left Shift
Right Shift
Unsigned Right Shift
Bitwise AND
Bitwise exclusive OR

C Operator Precedence and Associativity I
Introduction of Lexical Analyzer:
√ Lexical Analysis is the first phase of the compiler also known as a scanner.
It converts the High level input program into a sequence of Tokens.
√ The lexical analyzer is the part of the compiler that detects the token of
the program and sends it to the syntax analyzer.
√ Token is the smallest entity of the code, it is either a keyword, identifier,
constant, string literal, symbol. Examples of different types of tokens in C.

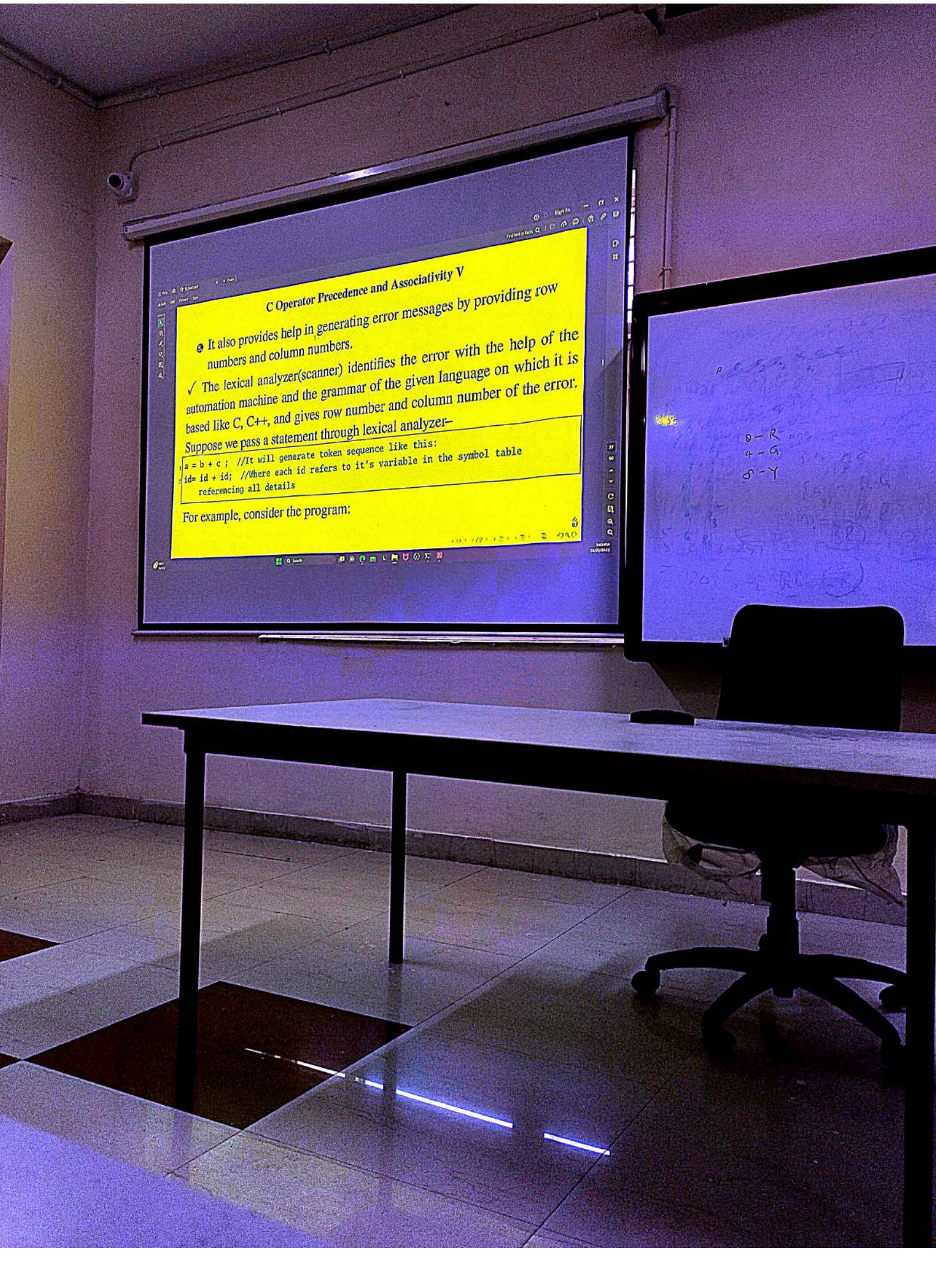

C Operator Precedence and Associativity III
• Type token (id, number, real, . . .)
• Punctuation tokens (IF, void, return, . . .)
• Alphabetic tokens (keywords)
• Keywords; Examples-for, while, if etc.
• Identifier; Examples-Variable name, function name, etc.
• Operators; Examples '+', '++', '-' etc.
• Separators; Examples ',' ';' etc
Example of Non-Tokens:
Comments, preprocessor directive, macros, blanks, tabs, newline, etc.

C Operator Precedence and Associativity IV

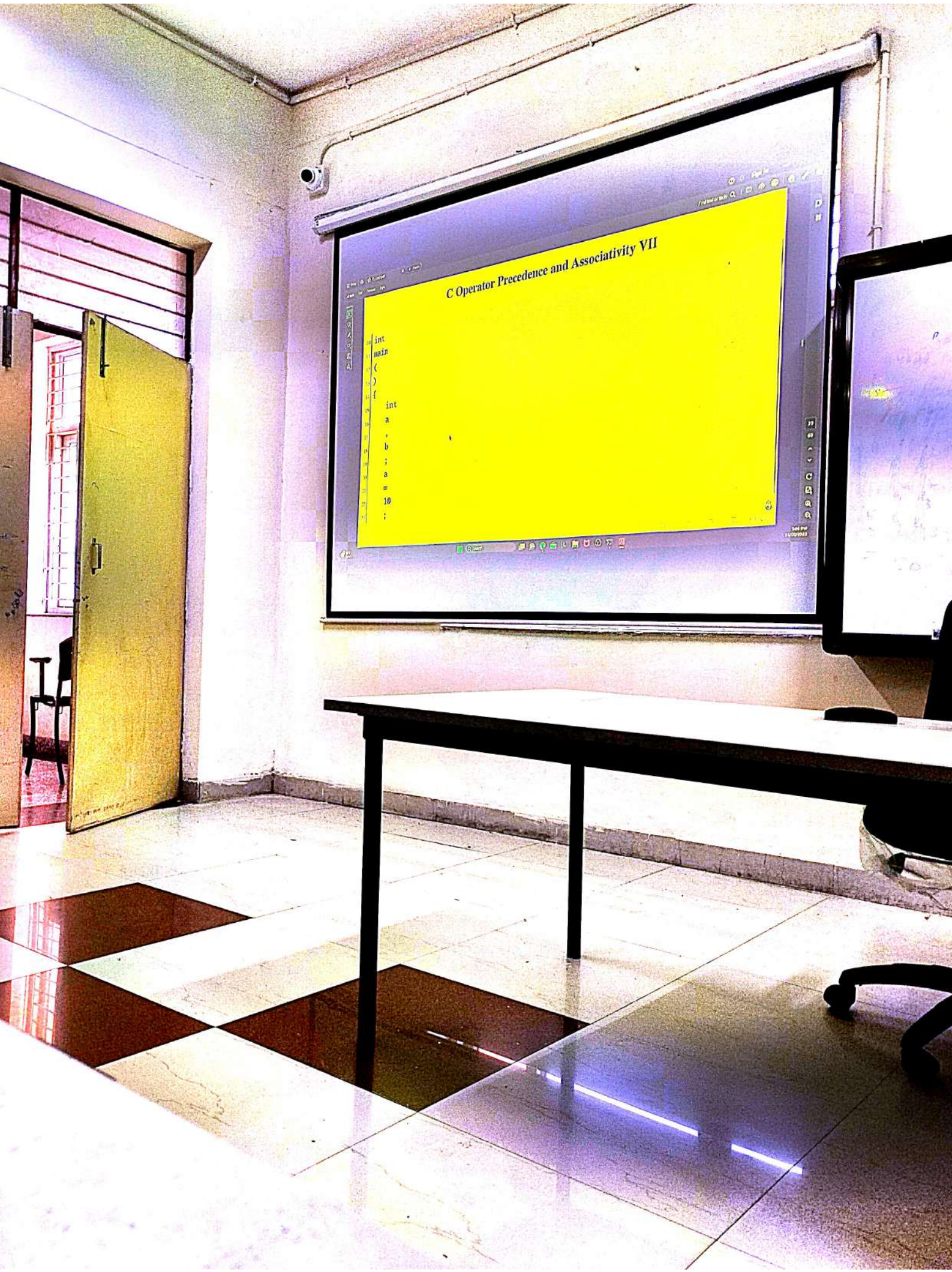
Lexeme: The sequence of characters matched by a pattern to form the corresponding token or a sequence of input characters that comprises a single token is called a lexeme. eg- "float", "abs_zero_Kelvin", "=", ";", "273", ";".

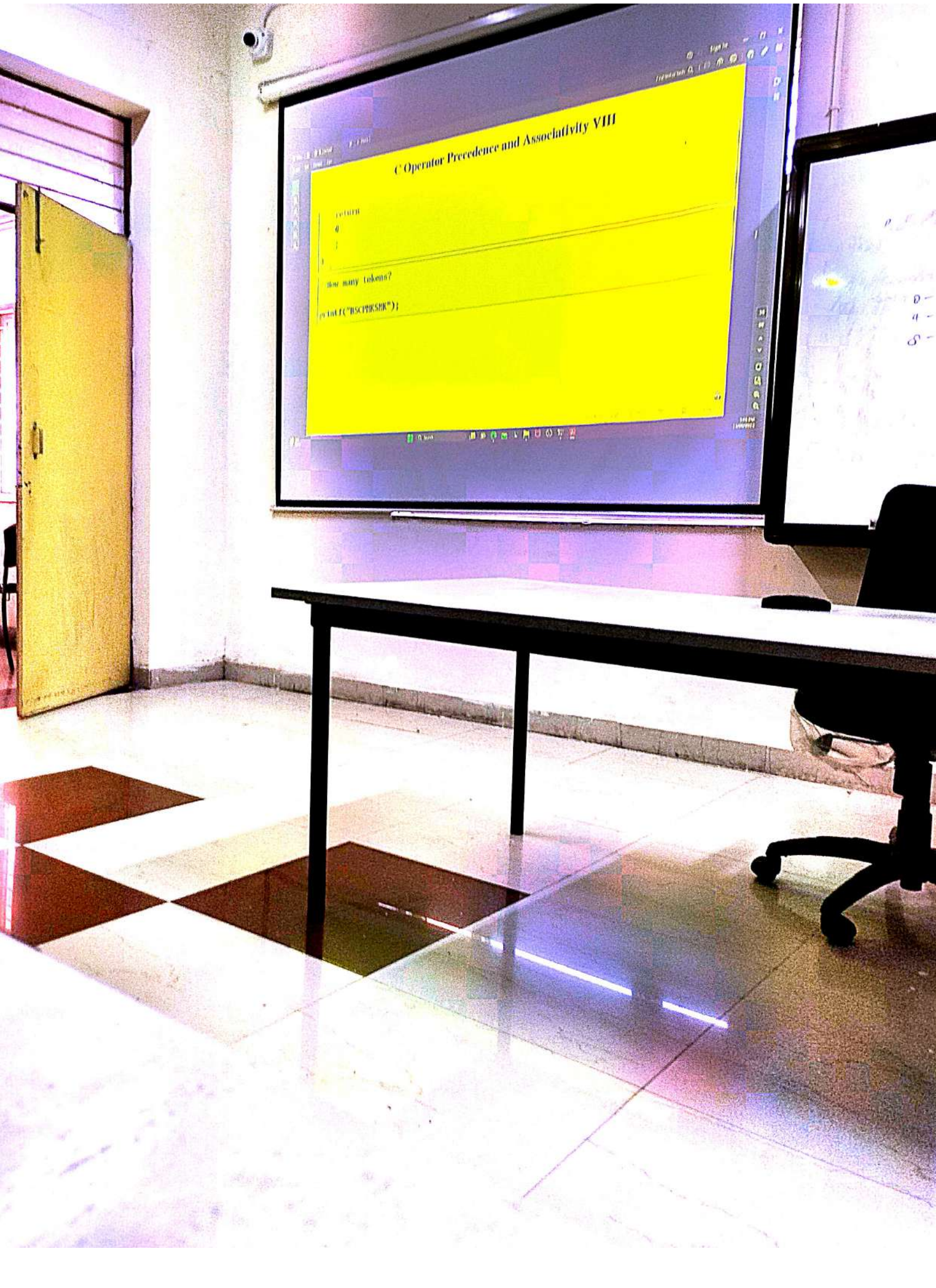
How Lexical Analyzer functions

1. It always matches the longest character sequence.
2. Tokenization i.e. Dividing the program into valid tokens.
3. Remove white space characters.
4. Remove comments.


C Operator Precedence and Associativity V
It also provides help in generating error messages by providing row numbers and column numbers.
✓ The lexical analyzer(scanner) identifies the error with the help of the automation machine and the grammar of the given language on which it is based like C, C++, and gives row number and column number of the error.
Suppose we pass a statement through lexical analyzer–
a = b + c ; //It will generate token sequence like this:
id= id + id; //Where each id refers to it's variable in the symbol table referencing all details
For example, consider the program:


C Operator Precedence and Associativity VI
int a=5; // int a = 5 ; just for understanding.
Tokens:
int a = 5 ;
Another example:
int main()
{
// 2 variables
int a, b;
a = 10;
return 0;
}
Tokens:
R - R
G - G
Y - Y


C Operator Precedence and Associativity VII
int
main
(
)
{
int
a
,
b
;
a
=
10
;


C Operator Precedence and Associativity VIII
return
0
;
}
How many tokens?
printf("BSCPBKSBK");


C Operator Precedence and Associativity IX
int main()
{
int a = 10, b = 20;
printf("sum is :%d",a+b);
return 0;
}

//How many tokens?
int max(int i);

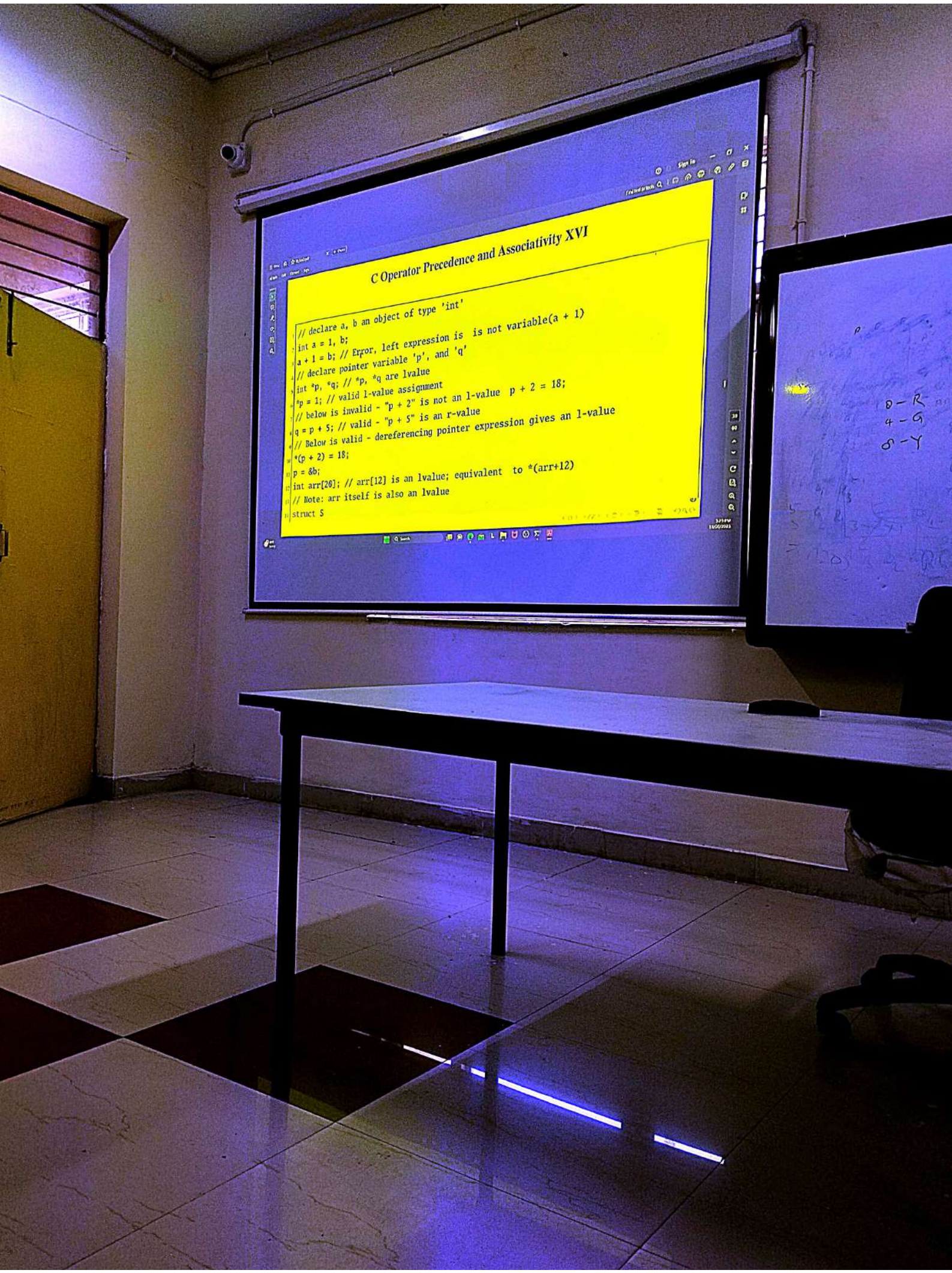
//Count number of tokens :


C Operator Precedence and Associativity X
• Lexical analyzer first read int and finds it to be valid and accepts as token
• max is read by it and found to be a valid function name after reading (token
• int is also a token , then again i as another token and finally ;
Answer: Total number of tokens 7:
int max (int i) { ;
//Count number of tokens :
printf("i = %d, &i = %x", i, &i);

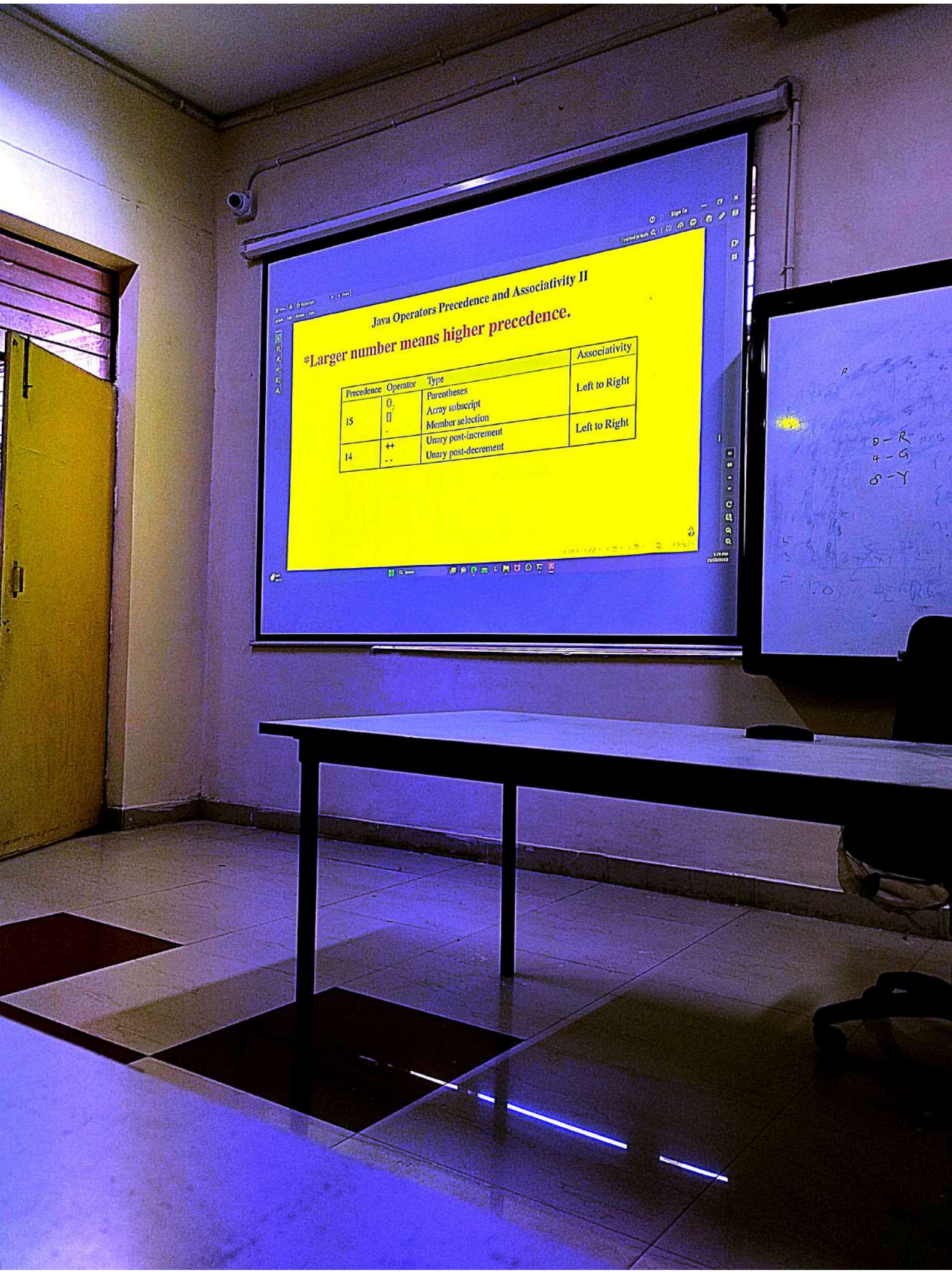

C Operator Precedence and Associativity XII
Lvalues and Rvalues in C:
There are two kinds of expressions in C -
lvalue - Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment operator (=).
✓ lvalue often represents as identifier.
lvalue(left value): simply means an object that has an identifiable location in memory (i.e. having an address).
-> In any assignment statement "lvalue" must have the capacity to hold the data.

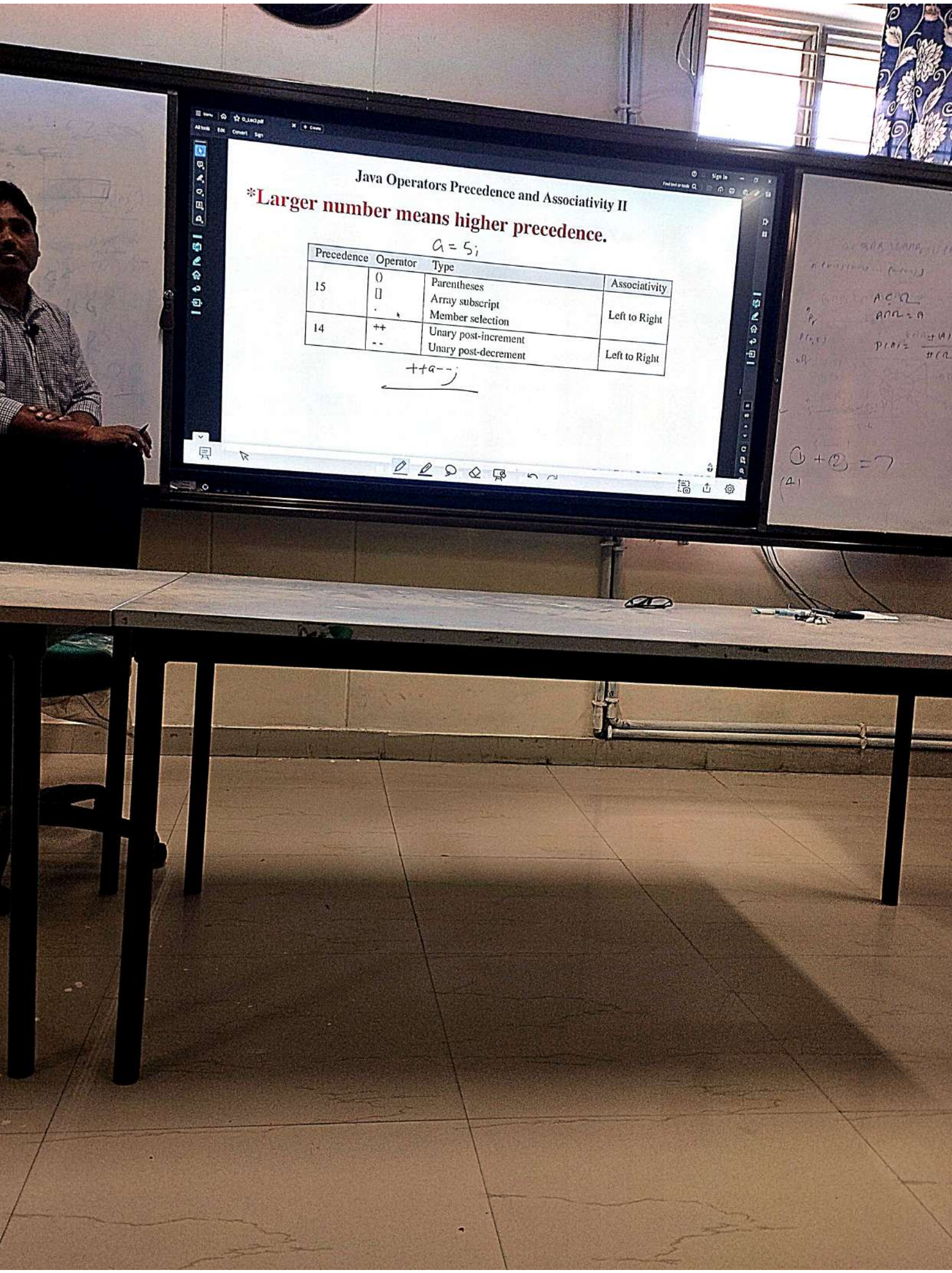

C Operator Precedence and Associativity XIII
-> lvalue must be a variable because they have the capability to store the data.
-> lvalue cannot be a function, expression (a+b) or a constant (like 3, 4 etc).
-> rvalue(right value): simply means an object that has no identifiable location in memory.
-> Anything which is capable of returning a constant expression or value.
-> Expression like (a+b) will return some constant value.
For example: a++; is equivalent to a = a+1;, here we have both lvalue and rvalue. before =, a is lvalue and after =, a+1 is rvalue.
Take our example a=b++; convert it into normal expression a=b=b + 1;


C Operator Precedence and Associativity XIV
Take our example (a=b)++; convert it into normal expression (a=b) = (a=b) + 1;
int g = 20; // valid statement
10 = 20; // invalid statement; would generate compile-time error.
// declare a an object of type 'int'
int a;
// a is an expression referring to an 'int' object as l-value
a = 1;
int b = a; // Ok, as l-value can appear on right
// Switch the operand around '=' operator
9 = a;
// Compilation error: as assignment is trying to change the value of assignment operator

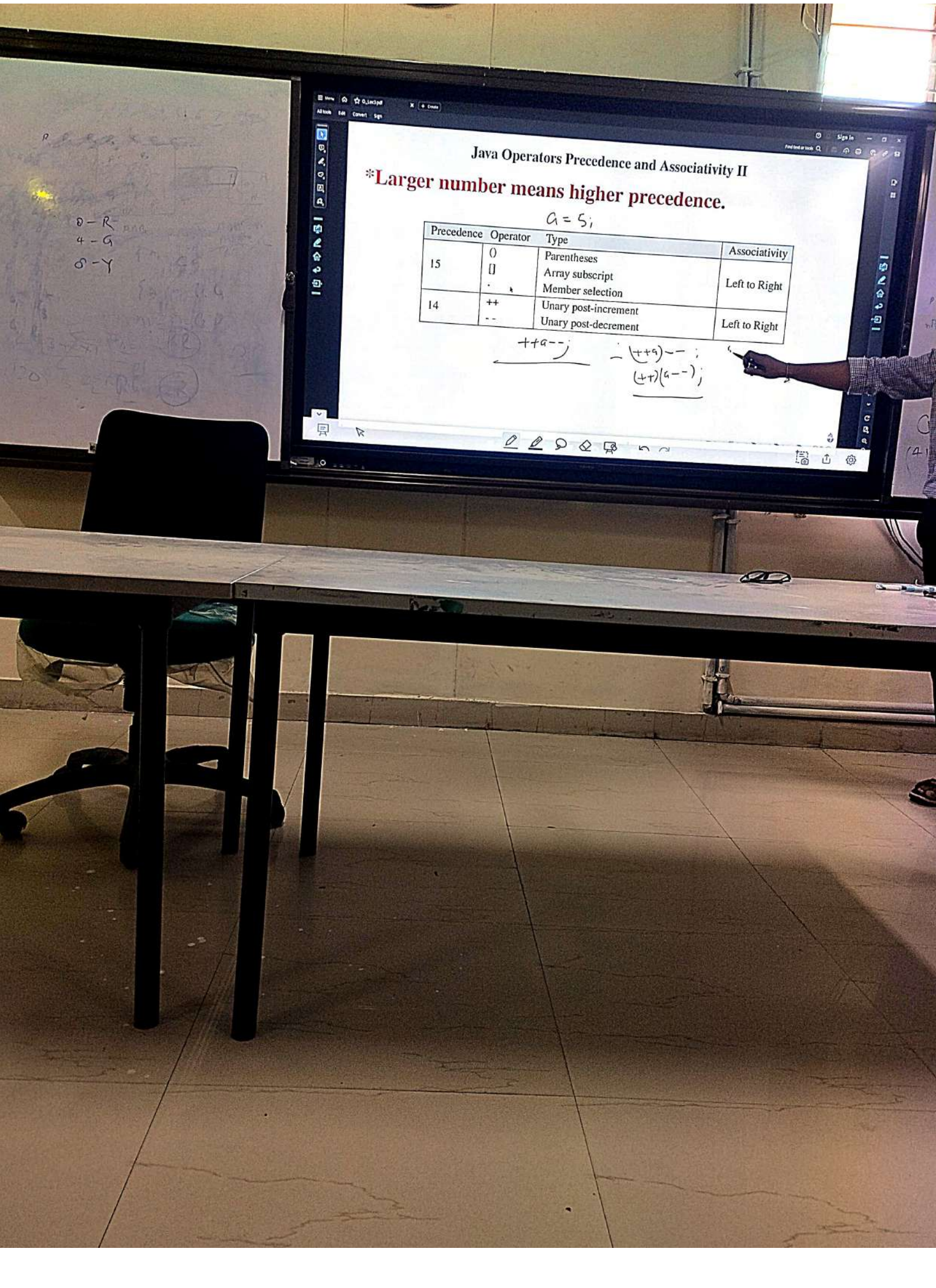

C Operator Precedence and Associativity XVI
// declare a, b an object of type 'int'
int a = 1, b;
a + 1 = b; // Error, left expression is is not variable(a + 1)
// declare pointer variable 'p', and 'q'
int *p, *q; // *p, *q are lvalue
*p = 1; // valid l-value assignment
// below is invalid - "p + 2" is not an l-value p + 2 = 18;
q = p + 5; // valid - "p + 5" is an r-value
// Below is valid - dereferencing pointer expression gives an l-value
*(p + 2) = 18;
p = &b;
int arr[20]; // arr[12] is an lvalue; equivalent to *(arr+12)
// Note: arr itself is also an lvalue
struct S


Java Operators Precedence and Associativity I
✓ Precedence of operators come into picture when in an expression we need to decide which operator will be evaluated first.
✓ Operator with higher precedence will be evaluated first.
int a=1;
int b=4;
int c;
// expression
c= a + b;
// Which one is correct
(c=a) + b or
c = (a+b)

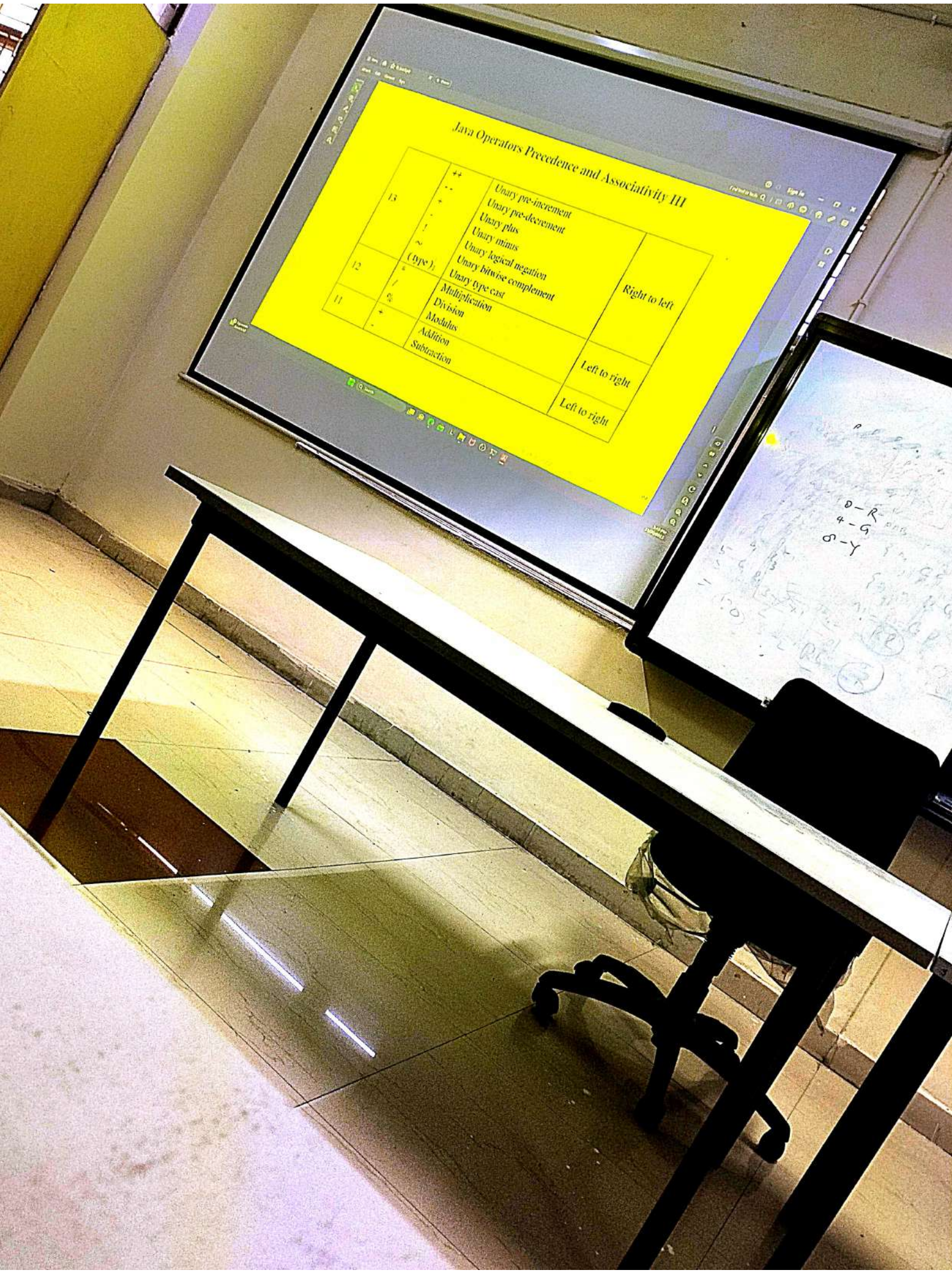

Java Operators Precedence and Associativity II
*Larger number means higher precedence.
Precedence
Operator
Type
Associativity
15
( )
[ ]
.
Parentheses
Array subscript
Member selection
Left to Right
14
++
--
Unary post-increment
Unary post-decrement
Left to Right



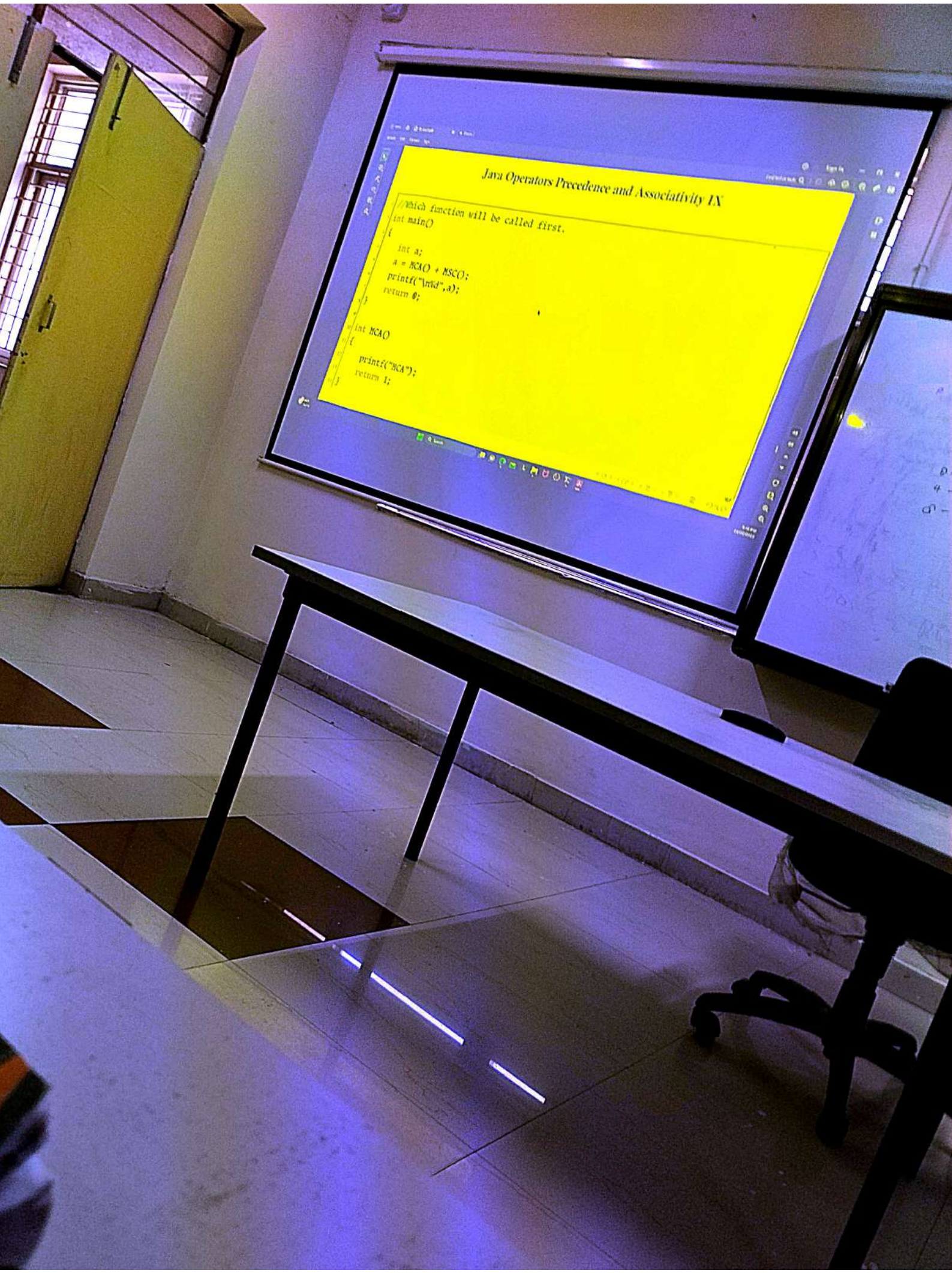
Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Left to Right

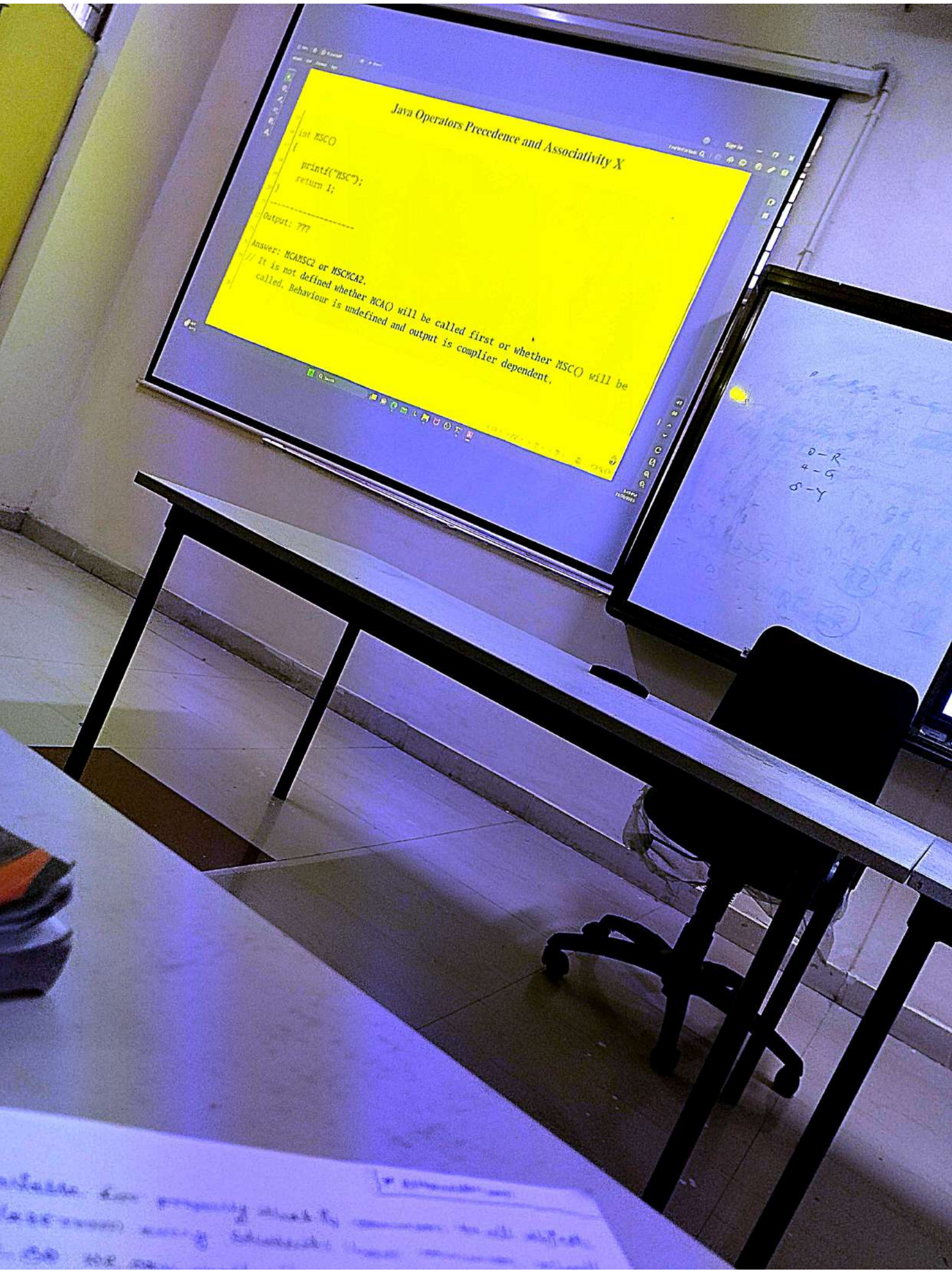


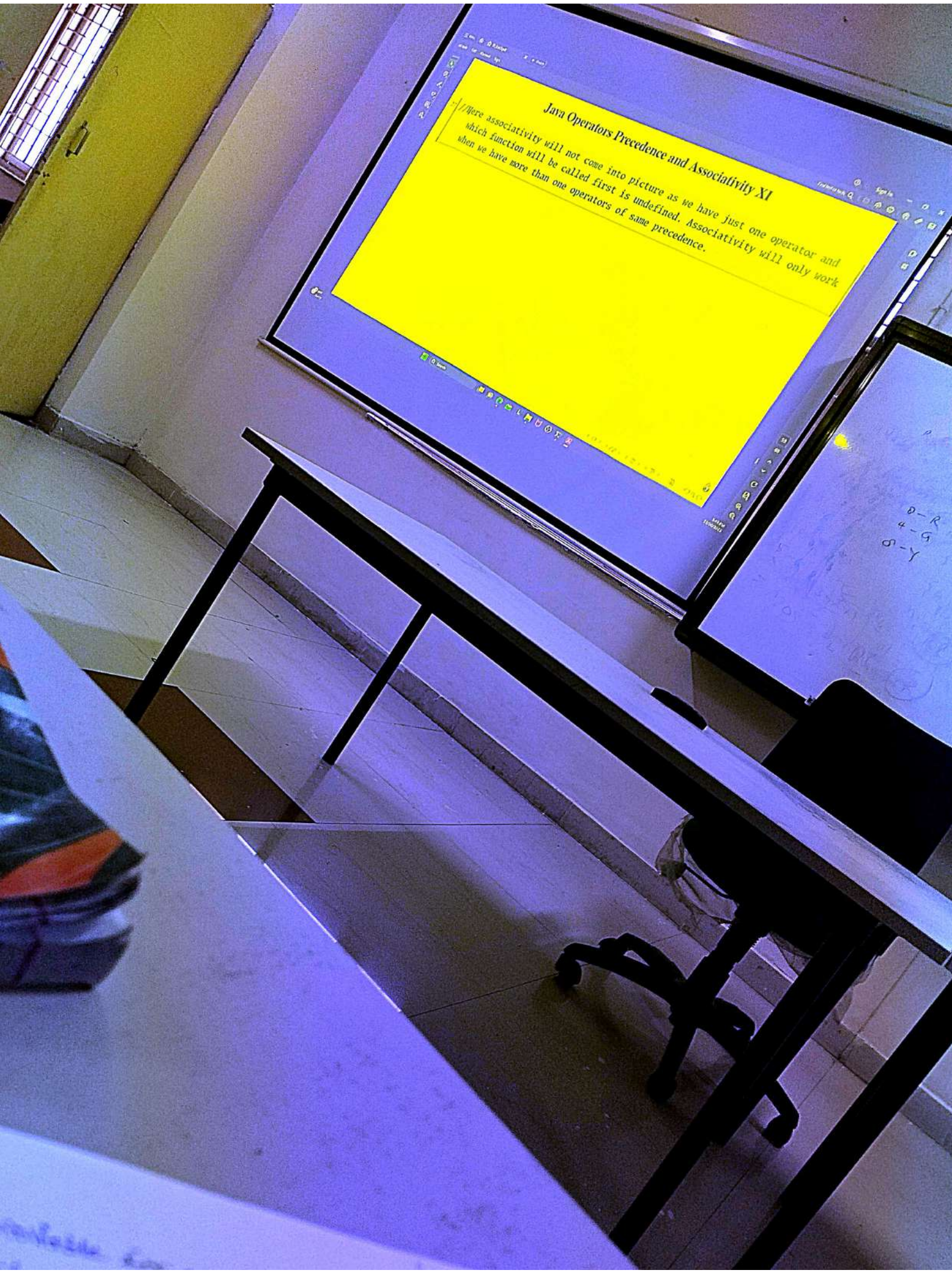
Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Left to Right

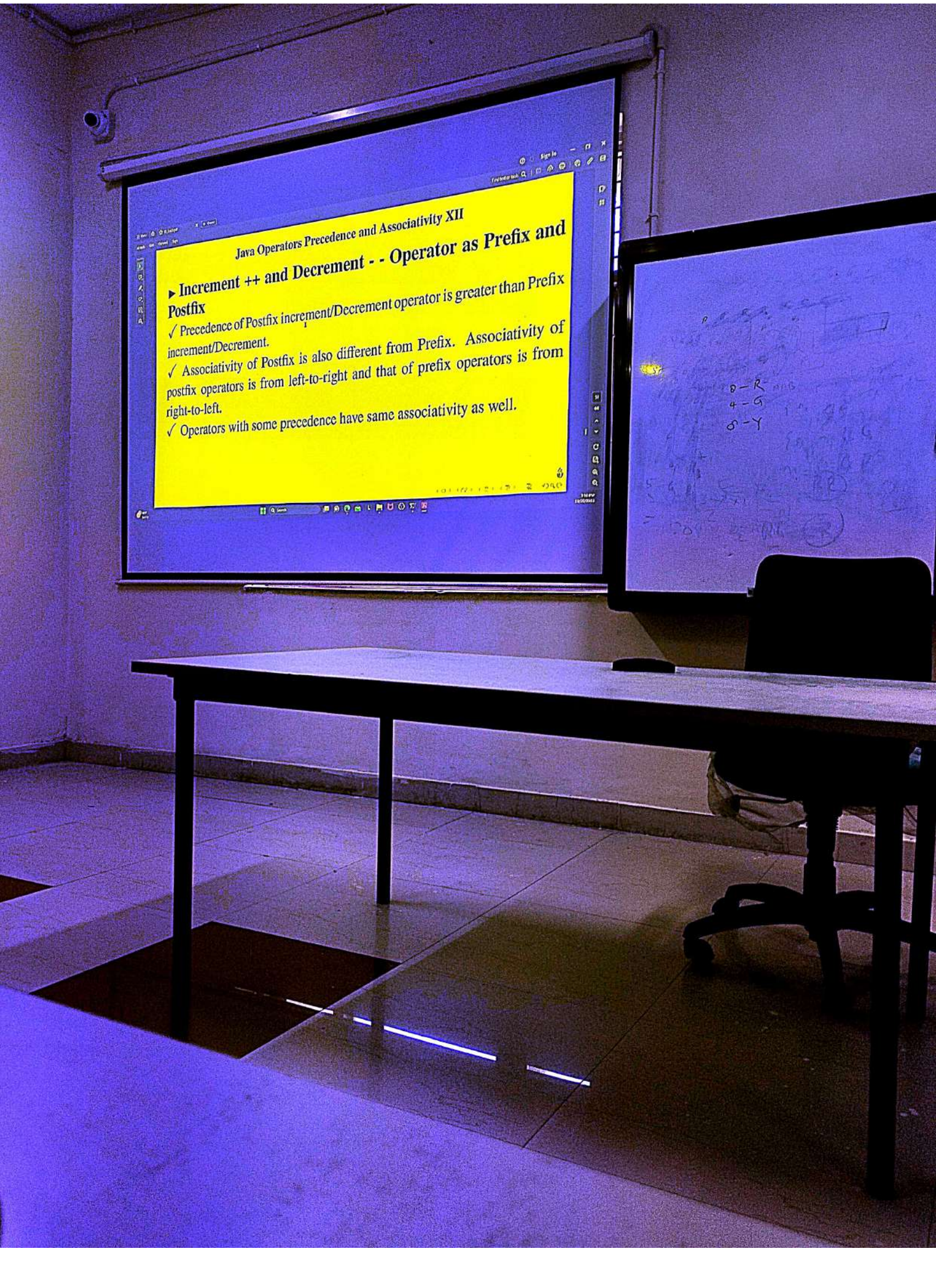

Java Operators Precedence and Associativity III
13
++
--
+
-
!
~
(type)
Unary pre-increment
Unary pre-decrement
Unary plus
Unary minus
Unary logical negation
Unary bitwise complement
Unary type cast
Right to left
12
*
/
%
Multiplication
Division
Modulus
Left to right
11
+
-
Addition
Subtraction
Left to right

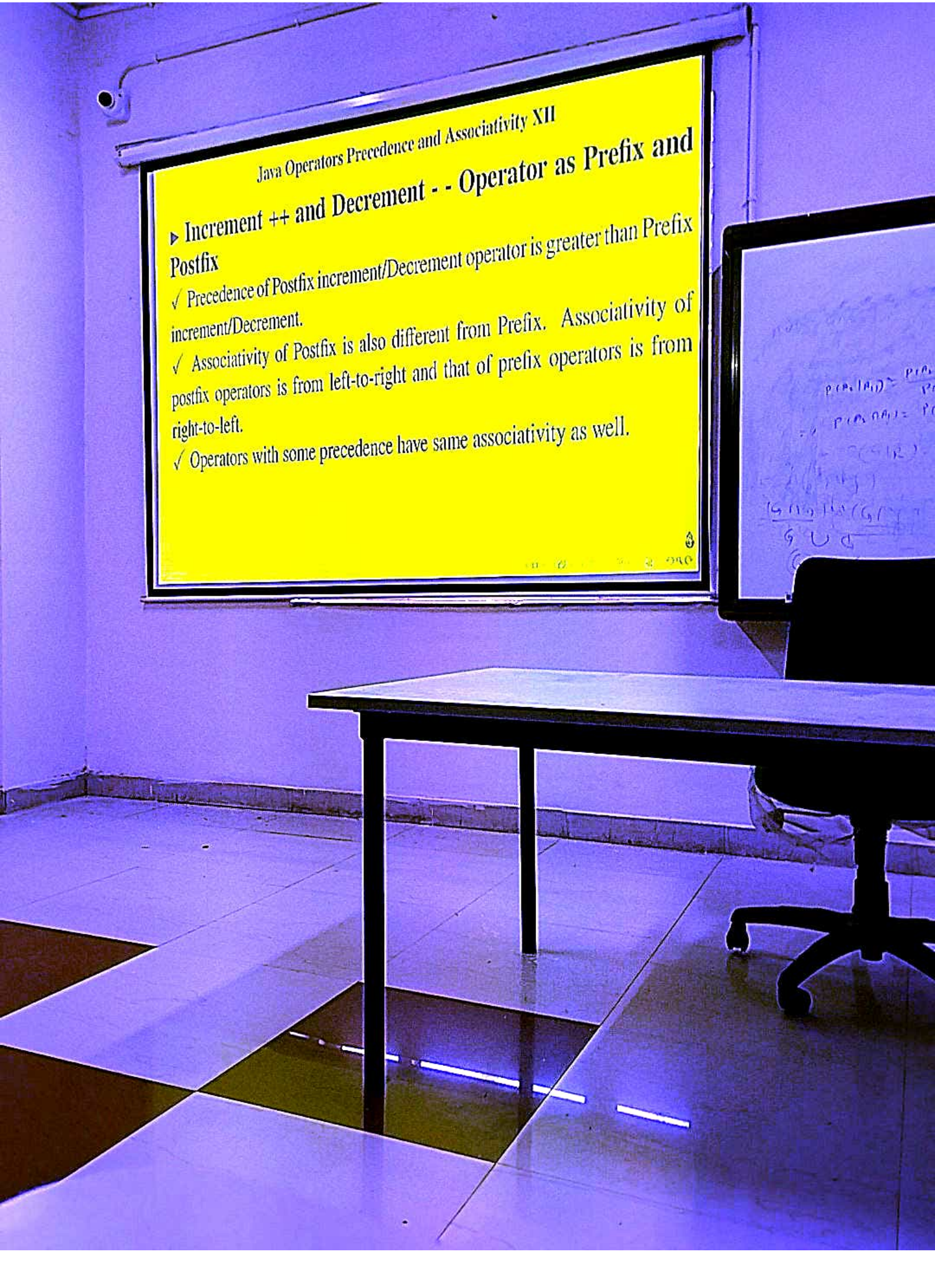

Java Operators Precedence and Associativity VIII
✓ ()- parenthesis in function calls.
✓ Parenthesis() operator having greater precedence than assignment(=) operator.
int val =fun();
// if suppose = operator is having greater precedence then, fun will belong to = operator and therefore it will be treated as a variable.
int (val = fun)();
// = operator is having less less precedence as compared to () therefore, () belongs to fun and will be treated as a function.
int val = (fun());

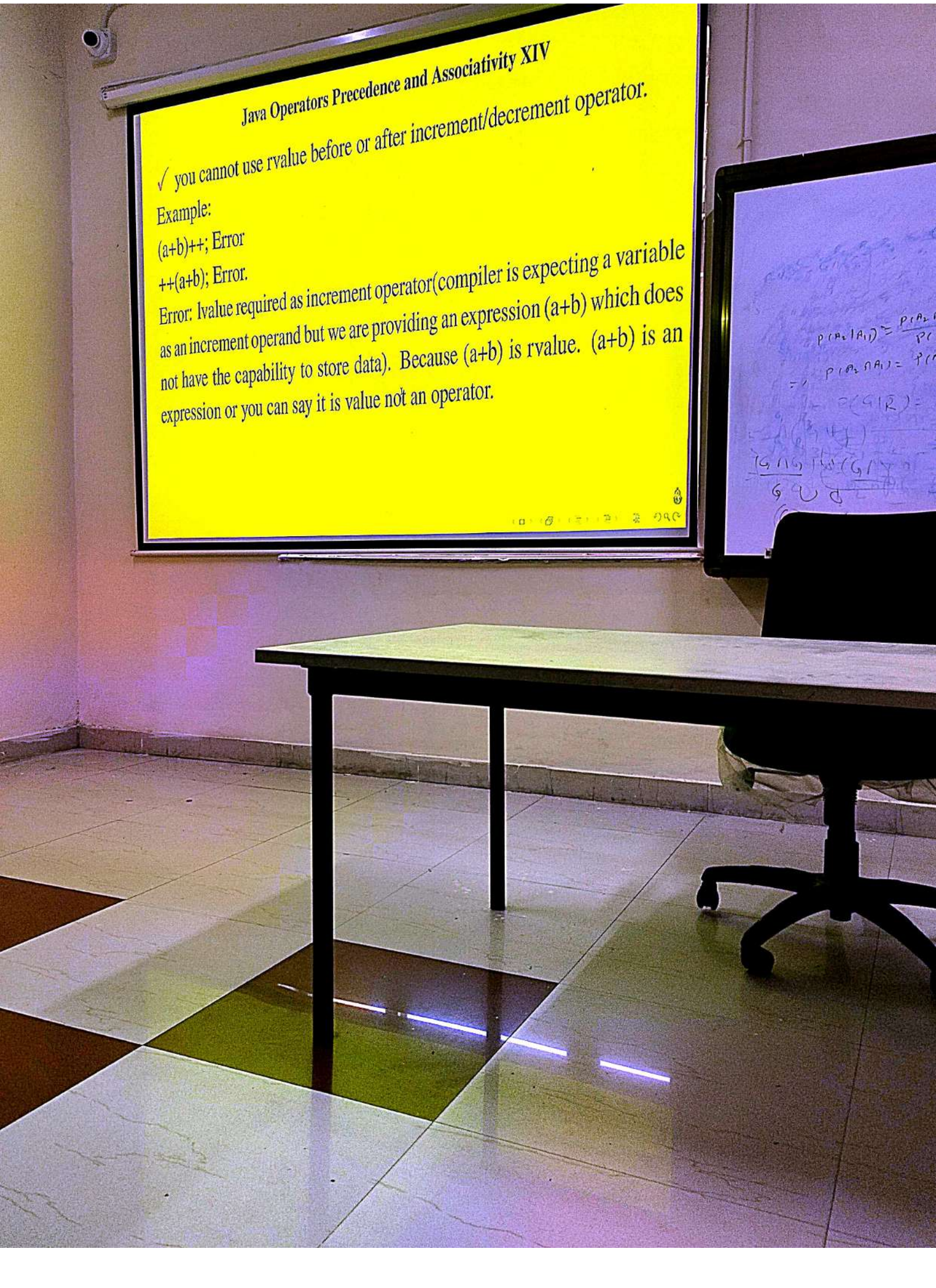

Java Operators Precedence and Associativity IX
//Which function will be called first.
int main()
{
int a;
a = MCA() + MSC();
printf("\n%d",a);
return 0;
}
int MCA()
{
printf("MCA");
return 1;
}


Java Operators Precedence and Associativity X
int MSC()
{
printf("MSC");
return 1;
}
Output: ???
Answer: MCAMSC2 or MSCMCA2.
// It is not defined whether MCA() will be called first or whether MSC() will be called. Behaviour is undefined and output is complier dependent.
D-R
4-G
S-Y


Java Operators Precedence and Associativity XI
//Here associativity will not come into picture as we have just one operator and
which function will be called first is undefined. Associativity will only work
when we have more than one operators of same precedence.


Java Operators Precedence and Associativity XII
▸ Increment ++ and Decrement - - Operator as Prefix and Postfix
✓ Precedence of Postfix increment/Decrement operator is greater than Prefix increment/Decrement.
✓ Associativity of Postfix is also different from Prefix. Associativity of postfix operators is from left-to-right and that of prefix operators is from right-to-left.
✓ Operators with some precedence have same associativity as well.

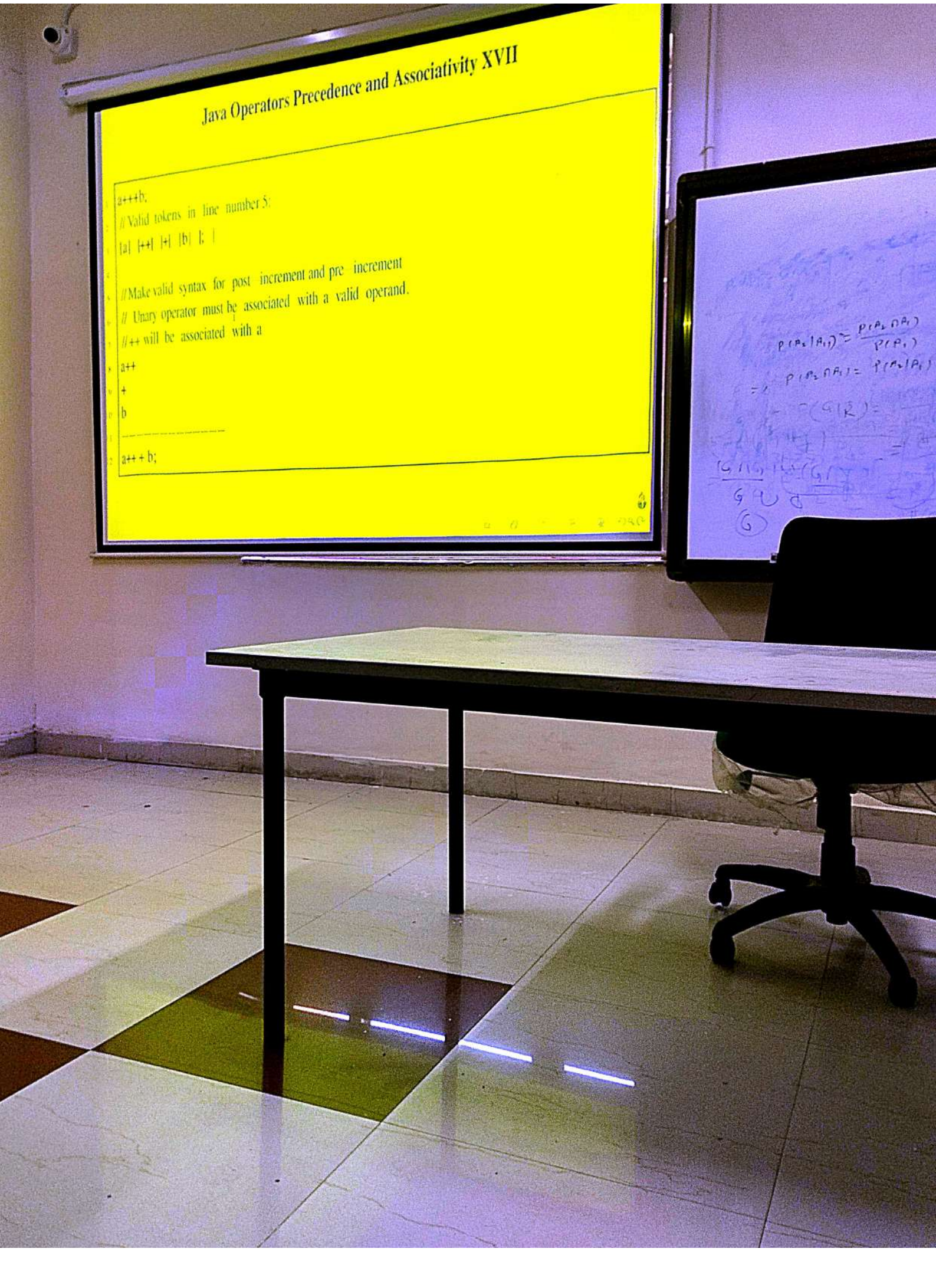

Java Operators Precedence and Associativity XII
Increment ++ and Decrement - - Operator as Prefix and Postfix
✓ Precedence of Postfix increment/Decrement operator is greater than Prefix increment/Decrement.
✓ Associativity of Postfix is also different from Prefix. Associativity of postfix operators is from left-to-right and that of prefix operators is from right-to-left.
✓ Operators with some precedence have same associativity as well.


Java Operators Precedence and Associativity XIV
✓ you cannot use rvalue before or after increment/decrement operator.
Example:
(a+b)++; Error
++(a+b); Error.
Error: lvalue required as increment operator(compiler is expecting a variable as an increment operand but we are providing an expression (a+b) which does not have the capability to store data). Because (a+b) is rvalue. (a+b) is an expression or you can say it is value not an operator.

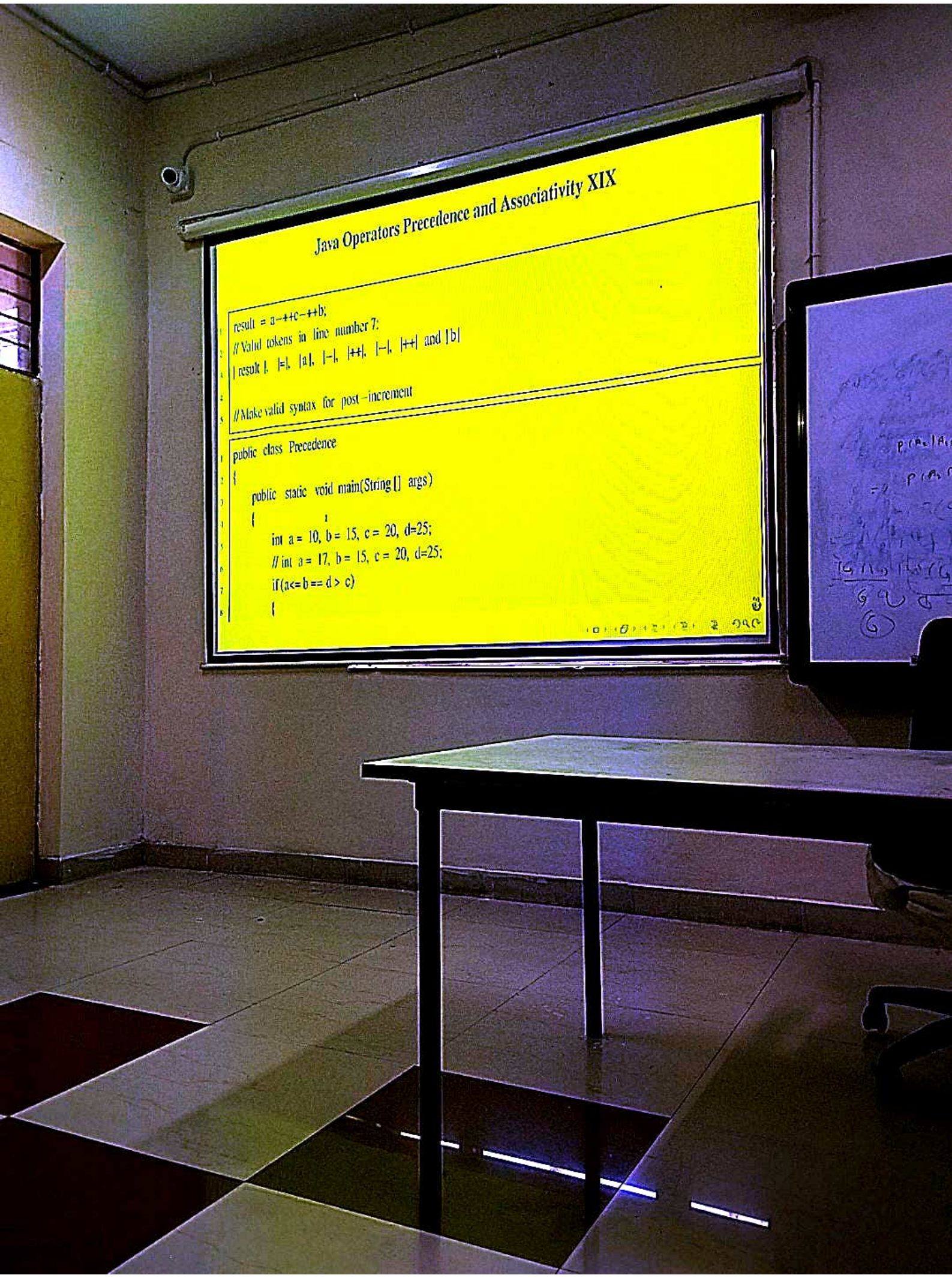

Java Operators Precedence and Associativity XV
int main()
{
int x = 1;
int y=0;
x=y++;
// (x=y) = (x=y) +1;
scanf("%d",&y);
printf("%d\n%d",x,y);
// return 0;
}
✓ Unary operator must be associated with a valid operand.


Java Operators Precedence and Associativity XV
int main()
{
int x = 1;
int y=0;
x=y++;
// (x=y) = (x=y) +1;
scanf("%d",&y);
printf("%d\n%d",x,y);
// return 0;
}
✓ Unary operator must be associated with a valid operand.

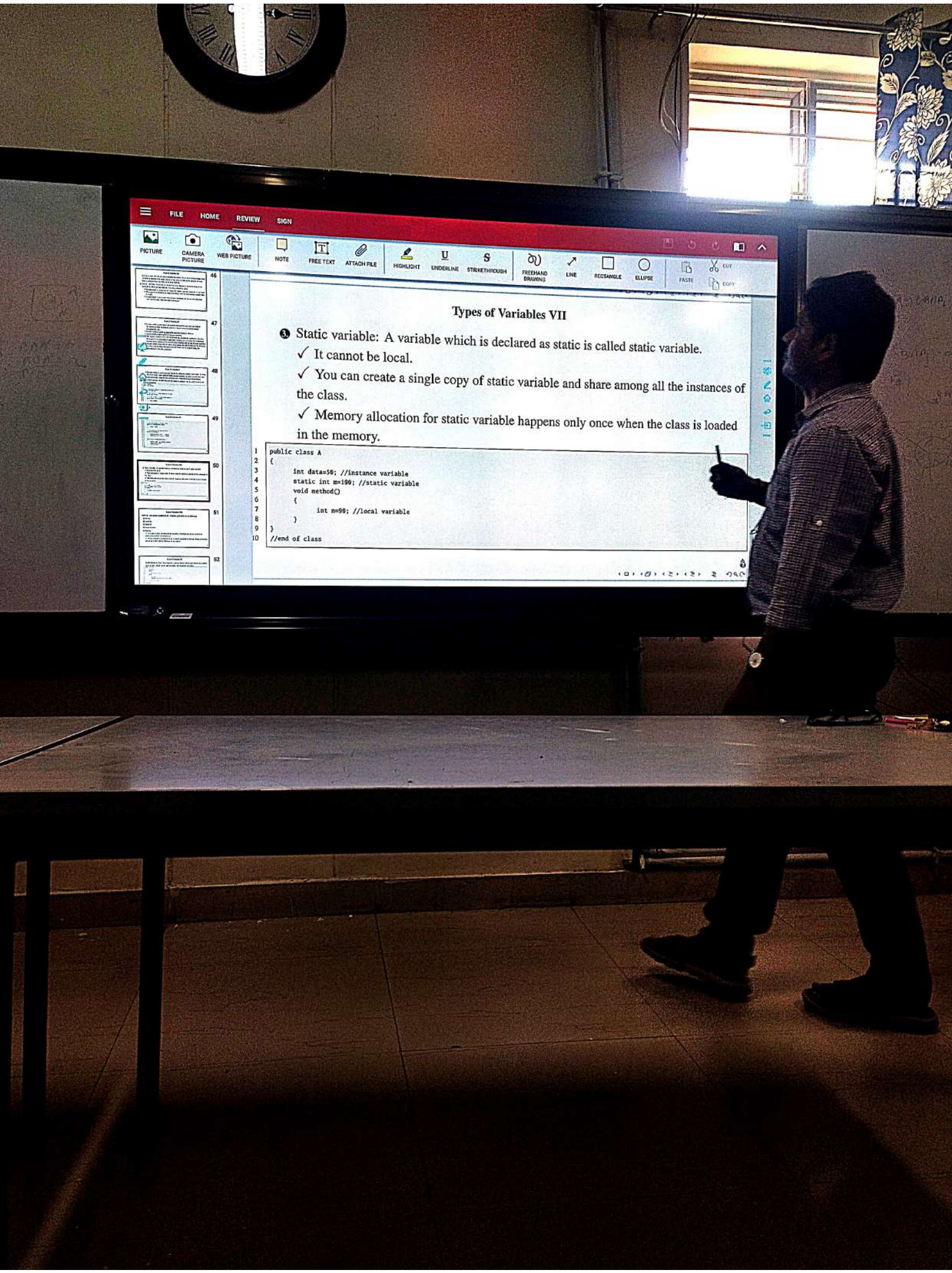

Java Operators Precedence and Associativity XVI
public class Precedence
{
public static void main(String[] args)
{
int a = 10, b = 5, c = 1;
System.out.println (a+++b);
System.out.println (a +++ b);
System.out.println (a++ + b);
System.out.println (a +++b);
System.out.println (a + ++b);
System.out.println (a+ ++b);
}
}

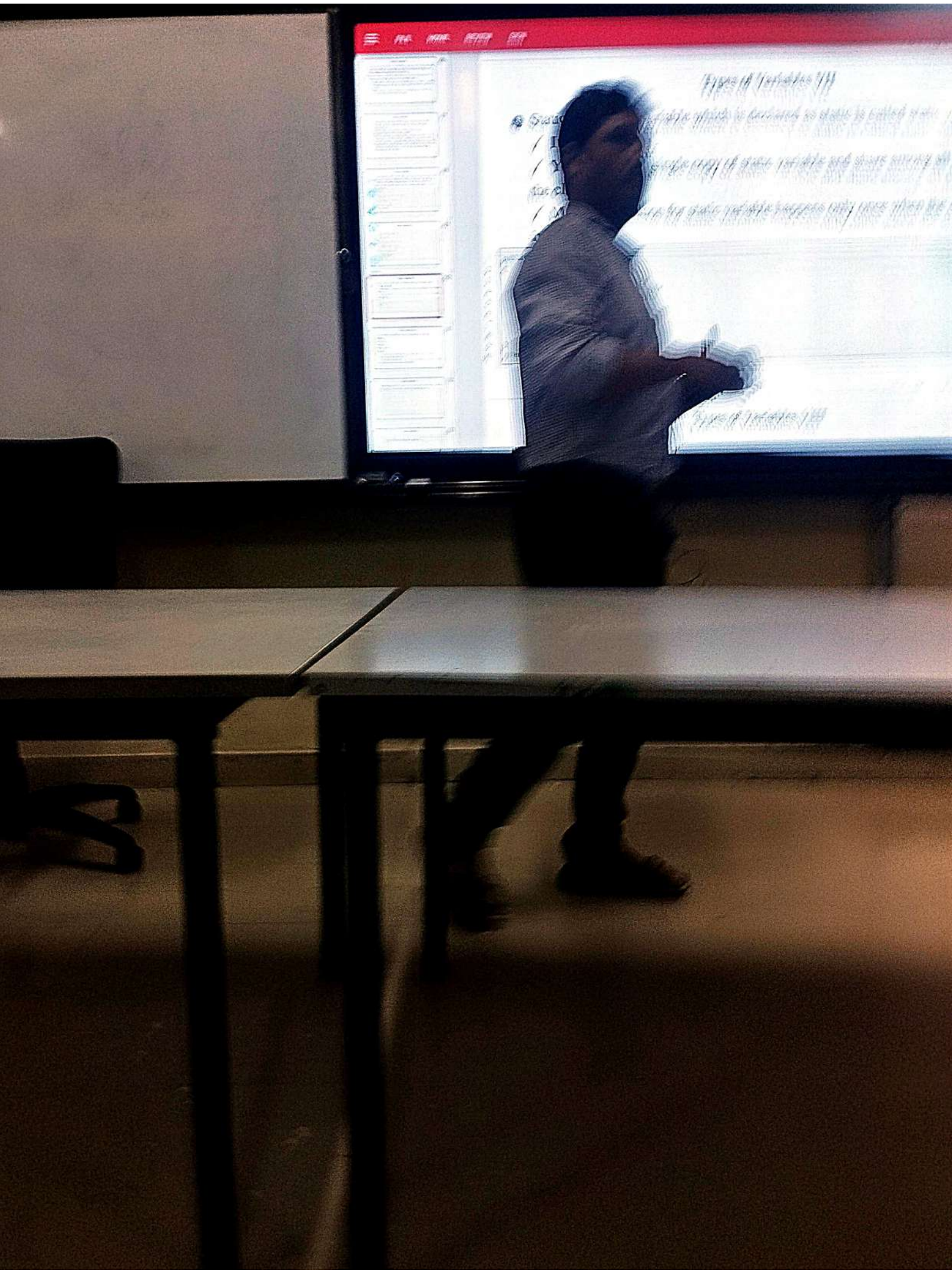

Java Operators Precedence and Associativity XVII
a+++b;
// Valid tokens in line number 5:
[a] [++] [+] [b] [;]
// Make valid syntax for post increment and pre increment
// Unary operator must be associated with a valid operand.
// ++ will be associated with a
a++
+
b
a++ + b;


Java Operators Precedence and Associativity XVIII
public class Precedence
{
public static void main(String [] args)
{
int a = 10, b = 5, c = 1, result ;
result = a--+c--+b;
System.out. println (result);
}
}


Java Operators Precedence and Associativity XIX
result = a-++c-++b;
// Valid tokens in line number 7:
| result |, |=|, |a|, |-|, |++|, |-|, |++| and |b|
// Make valid syntax for post-increment
public class Precedence
{
public static void main(String[] args)
{
int a = 10, b = 15, c = 20, d=25;
// int a = 17, b = 15, c = 20, d=25;
if(a<=b == d > c)
{


Java Operators Precedence and Associativity XXI
|a| |<| |=| |b| |=| |=| |d| |>| |c|
OR
|a| |<=| |b| |==| |d| |>| |c|
|<=| --> Precedence 9
|==| --> Precedence 8
|>| --> Precedence 9
((a<=b) == (d>c))
(1 == 1)


FILE
HOME
REVIEW
SIGN
PICTURE
CAMERA PICTURE
WEB PICTURE
NOTE
FREE TEXT
ATTACH FILE
HIGHLIGHT
UNDERLINE
STRIKETHROUGH
FREEHAND DRAWING
LINE
RECTANGLE
ELLIPSE
PASTE
CUT
COPY
Types of Variables VII
❸ Static variable: A variable which is declared as static is called static variable.
✓ It cannot be local.
✓ You can create a single copy of static variable and share among all the instances of the class.
✓ Memory allocation for static variable happens only once when the class is loaded in the memory.
public class A
{
int data=50; //instance variable
static int m=100; //static variable
void method()
{
int n=90; //local variable
}
}
//end of class



FILE HOME REVIEW SIGN

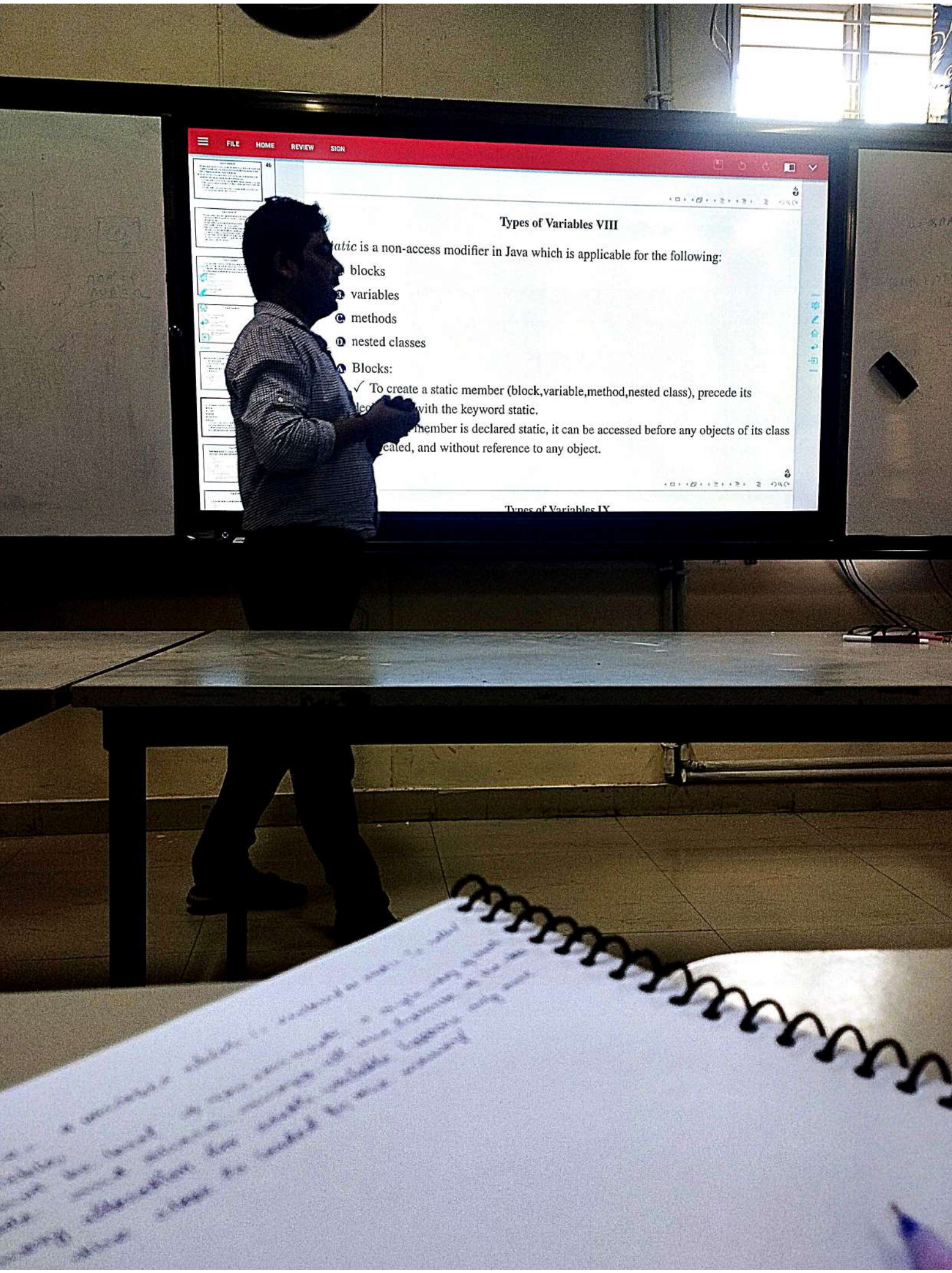
Types of Variables VII

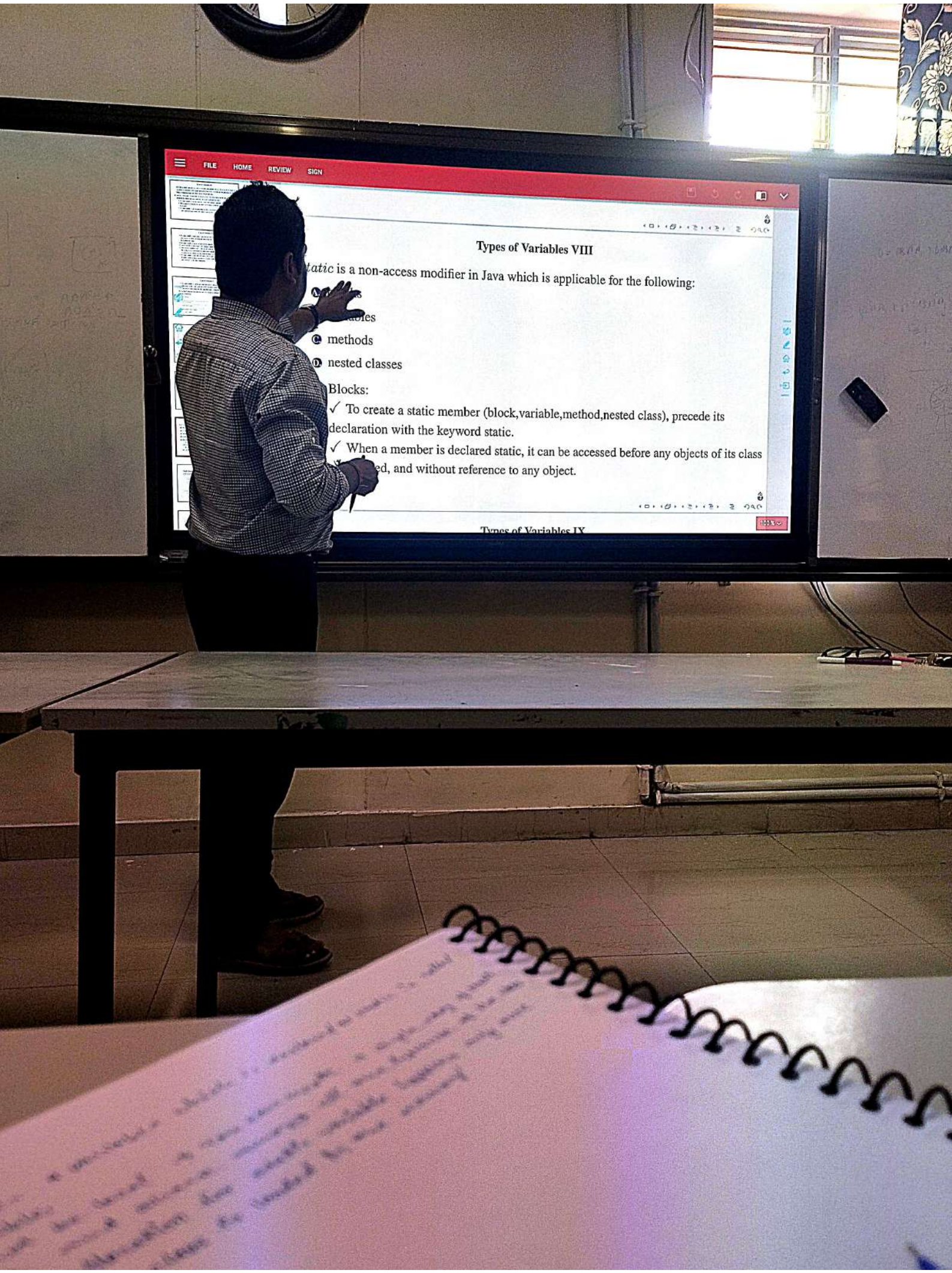
❸ Static variable: A variable which is declared as static is called static variable.

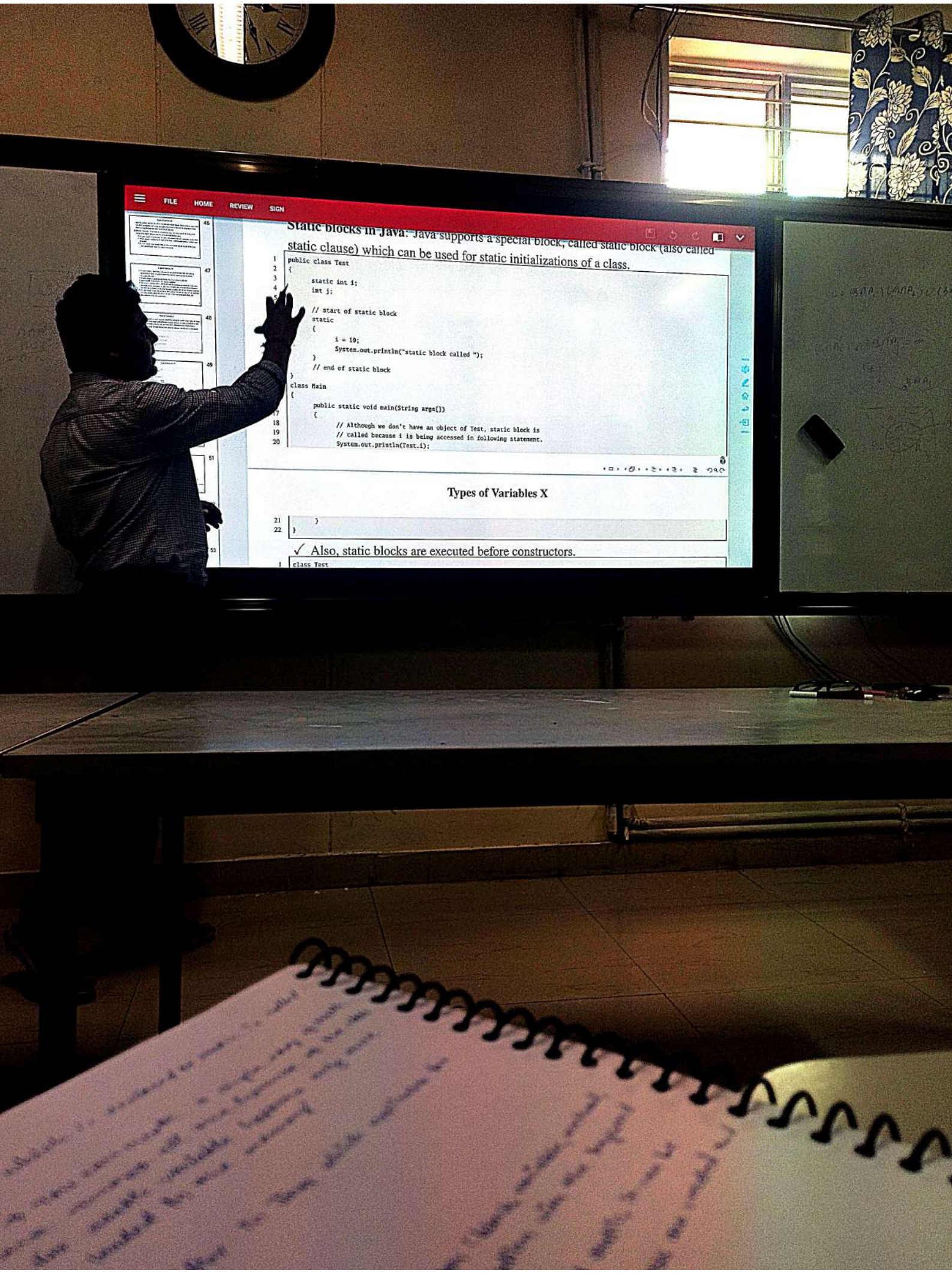
- ✓ It cannot be local.
- ✓ You can create a single copy of static variable and share among all the instances of the class.
- ✓ Memory allocation for static variable happens only once when the class is loaded in the memory.

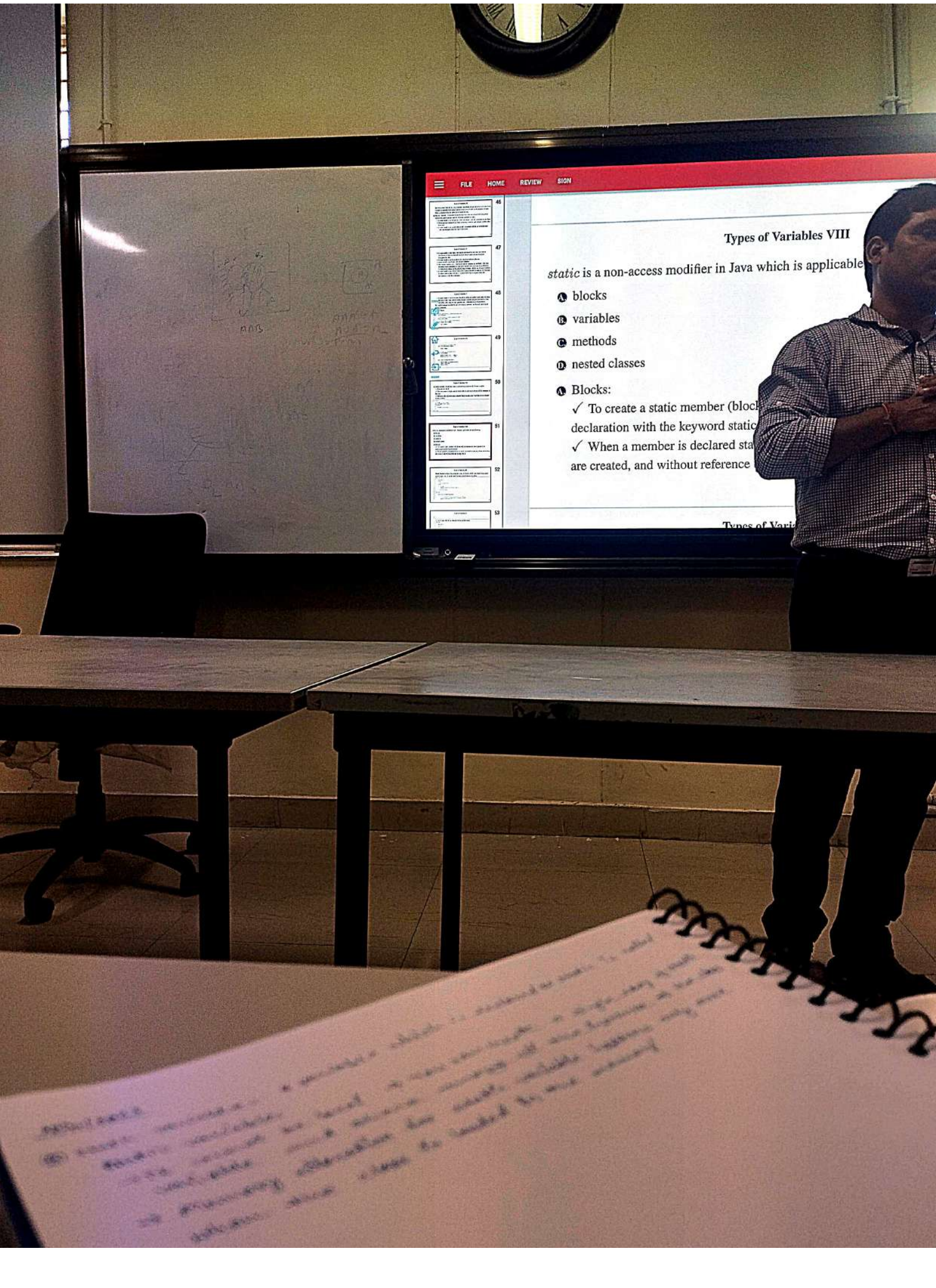
```
public class A
{
        int data=50; //instance variable
        static int m=100; //static variable
        void method()
        {
                int n=90; //local variable
        }
}
//end of class
```

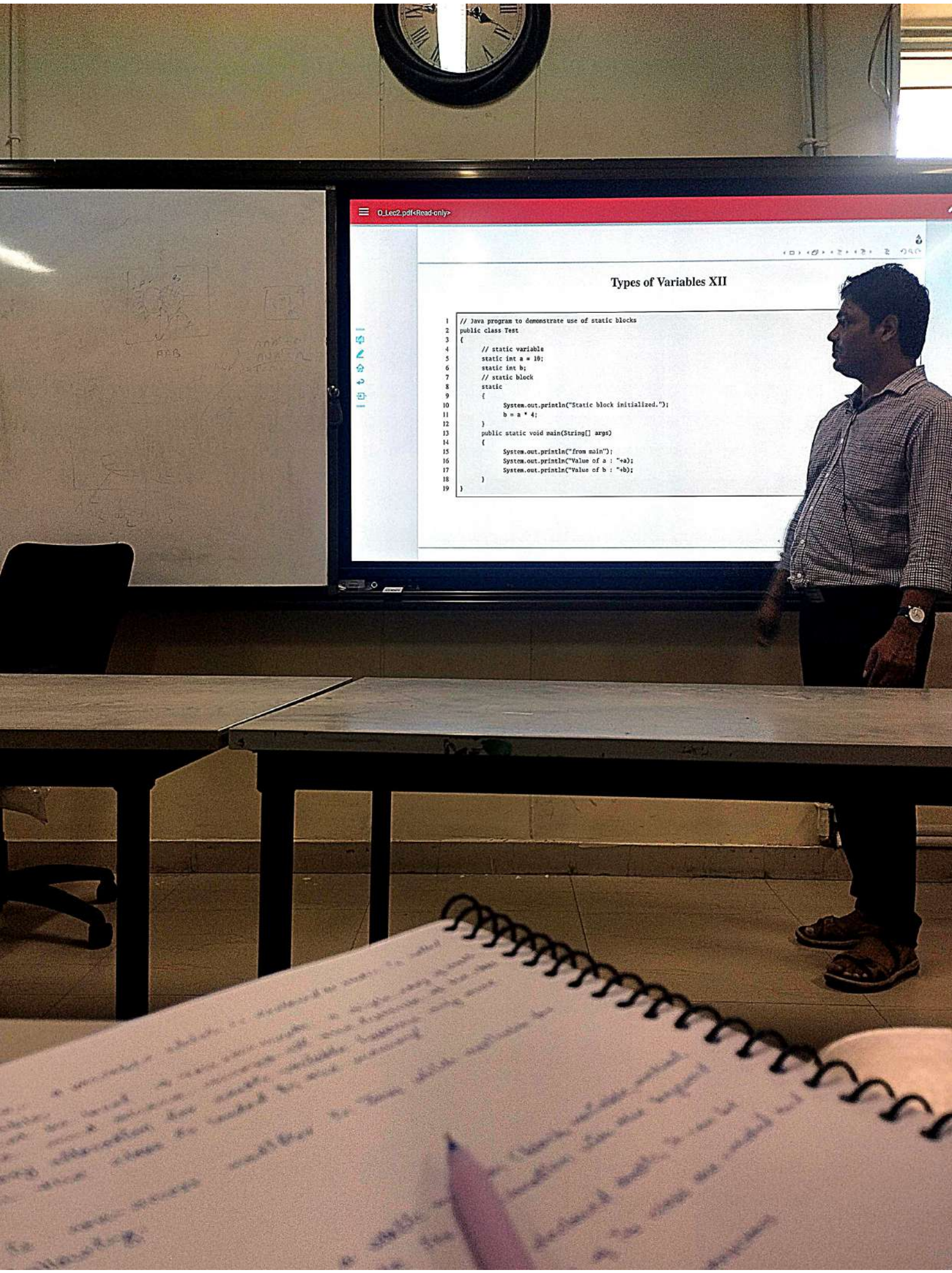
Types of Variables VIII

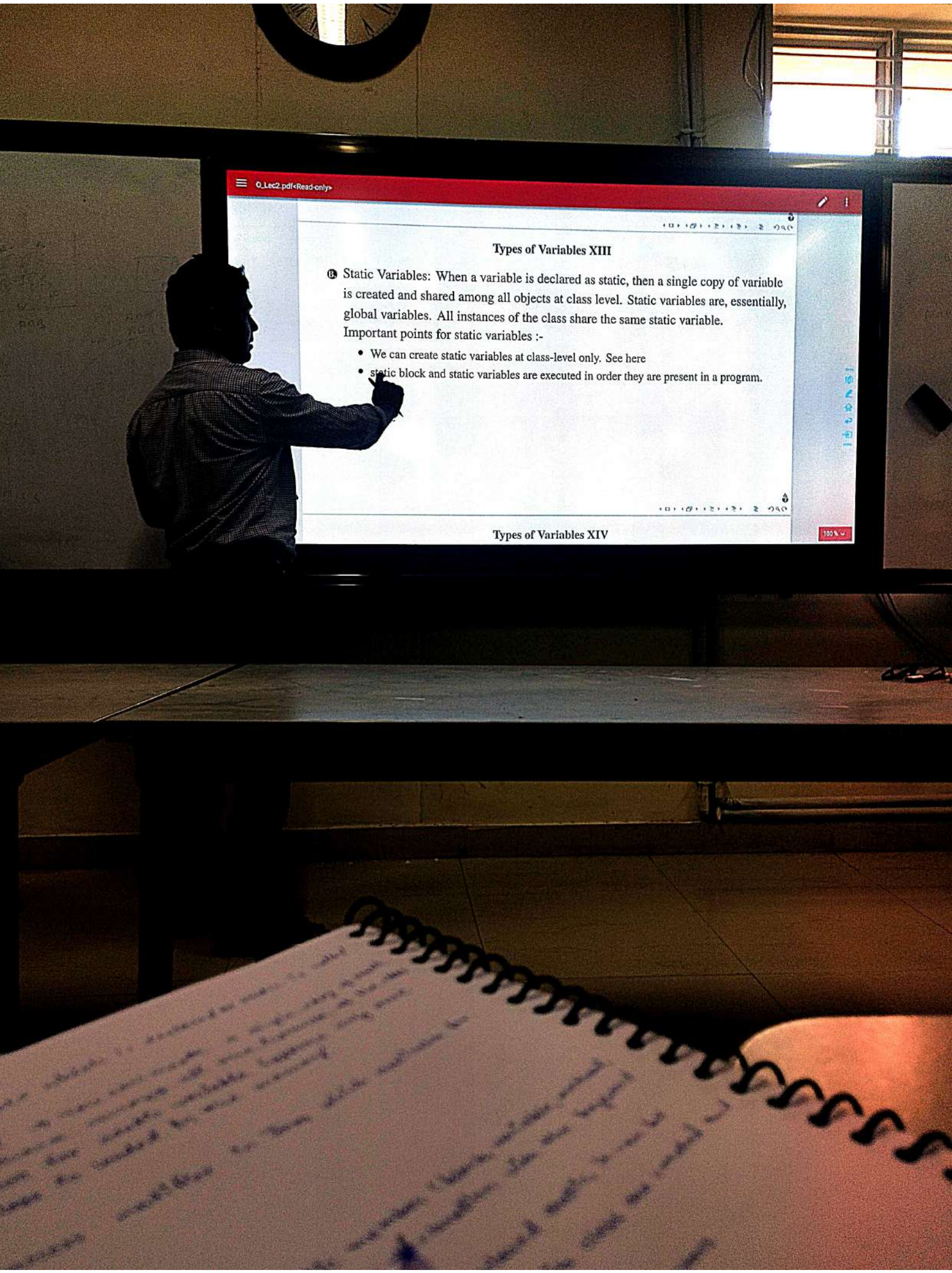

FILE
HOME
REVIEW
SIGN
Types of Variables VIII
atic is a non-access modifier in Java which is applicable for the following:
blocks
variables
methods
nested classes
Blocks:
✓ To create a static member (block,variable,method,nested class), precede its
with the keyword static.
member is declared static, it can be accessed before any objects of its class
caled, and without reference to any object.
Types of Variables IX

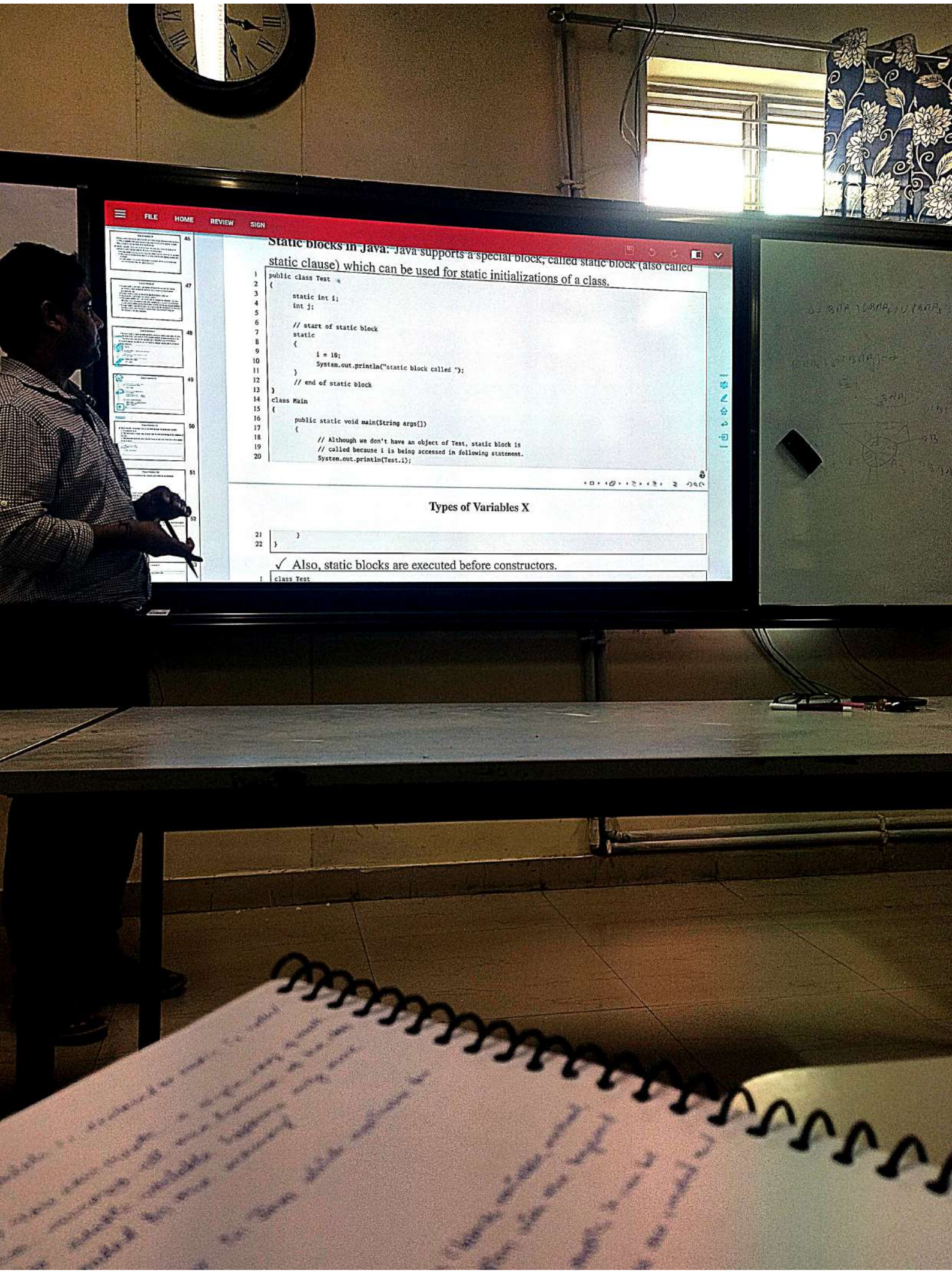

FILE
HOME
REVIEW
SIGN
Types of Variables VIII
tatic is a non-access modifier in Java which is applicable for the following:
methods
nested classes
Blocks:
✓ To create a static member (block,variable,method,nested class), precede its declaration with the keyword static.
✓ When a member is declared static, it can be accessed before any objects of its class
ed, and without reference to any object.
Types of Variables IX

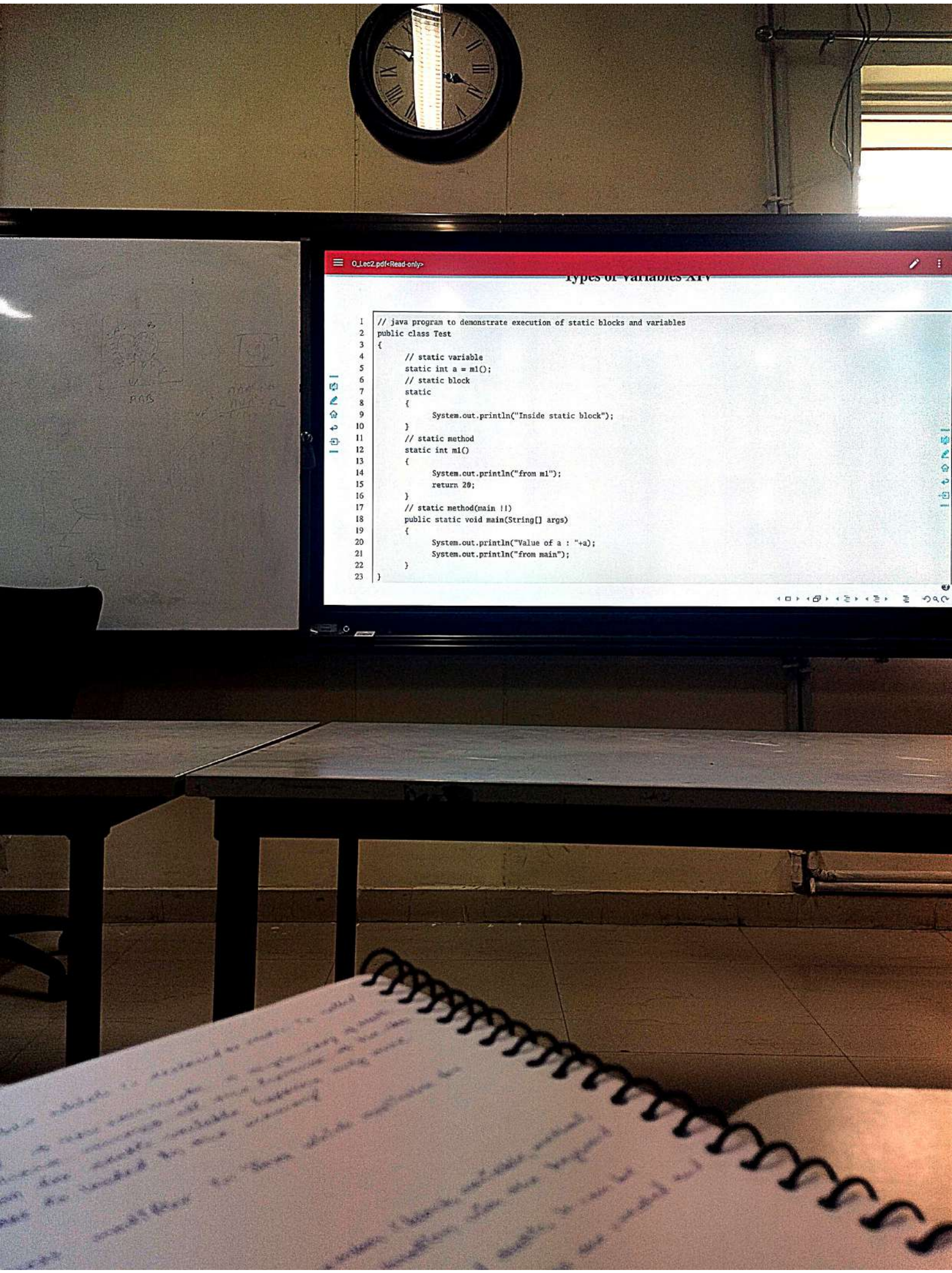

FILE
HOME
REVIEW
SIGN
Static blocks in Java: Java supports a special block, called static block (also called static clause) which can be used for static initializations of a class.
public class Test
{
static int i;
int j;
// start of static block
static
{
i = 10;
System.out.println("static block called ");
}
// end of static block
}
class Main
{
public static void main(String args[])
{
// Although we don't have an object of Test, static block is
// called because i is being accessed in following statement.
System.out.println(Test.i);
}
}
Types of Variables X
✓ Also, static blocks are executed before constructors.
class Test

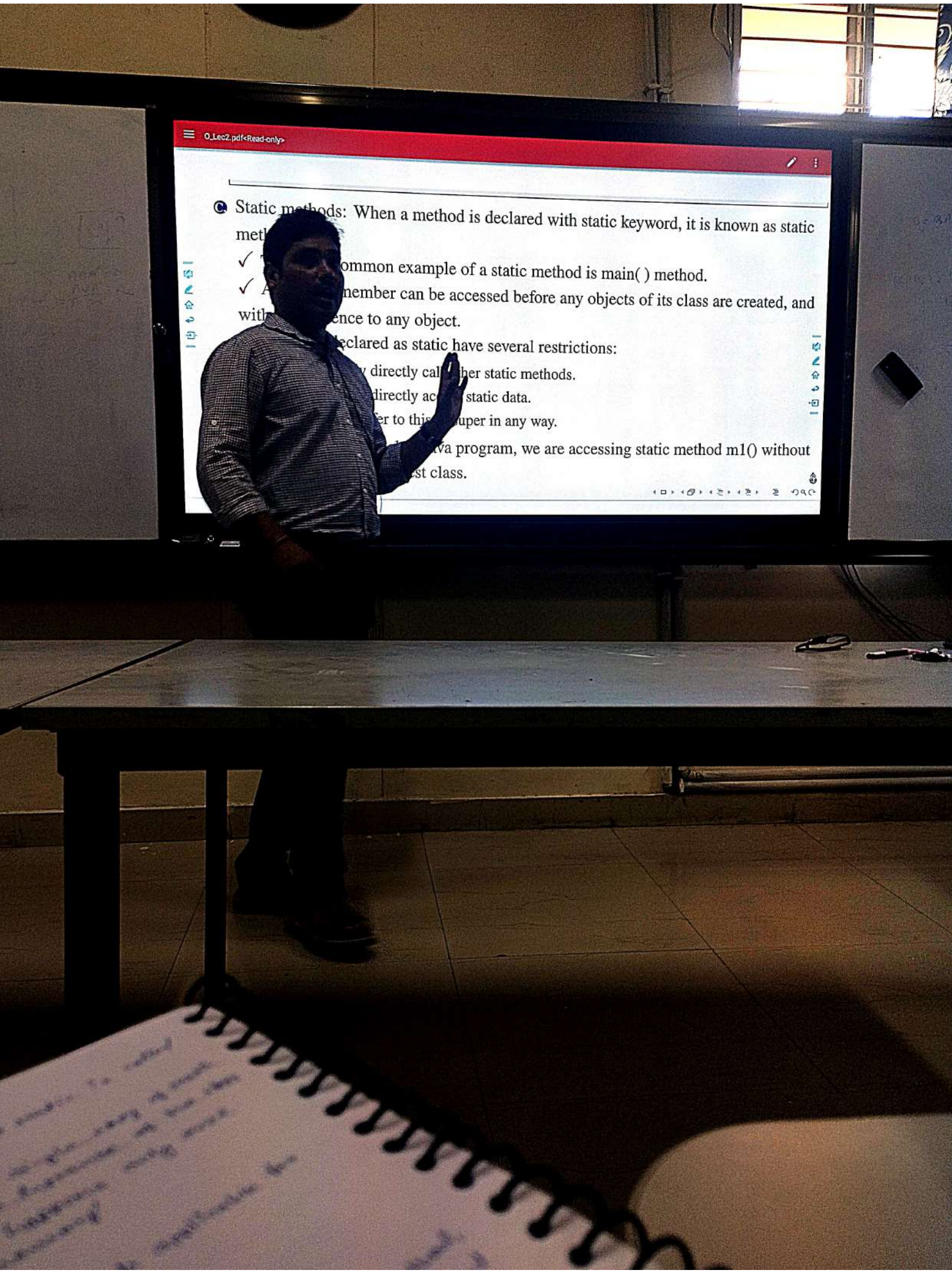

FILE
HOME
REVIEW
SIGN
Types of Variables VIII
static is a non-access modifier in Java which is applicable
A blocks
B variables
C methods
D nested classes
A Blocks:
✓ To create a static member (bloc
declaration with the keyword static
✓ When a member is declared sta
are created, and without reference

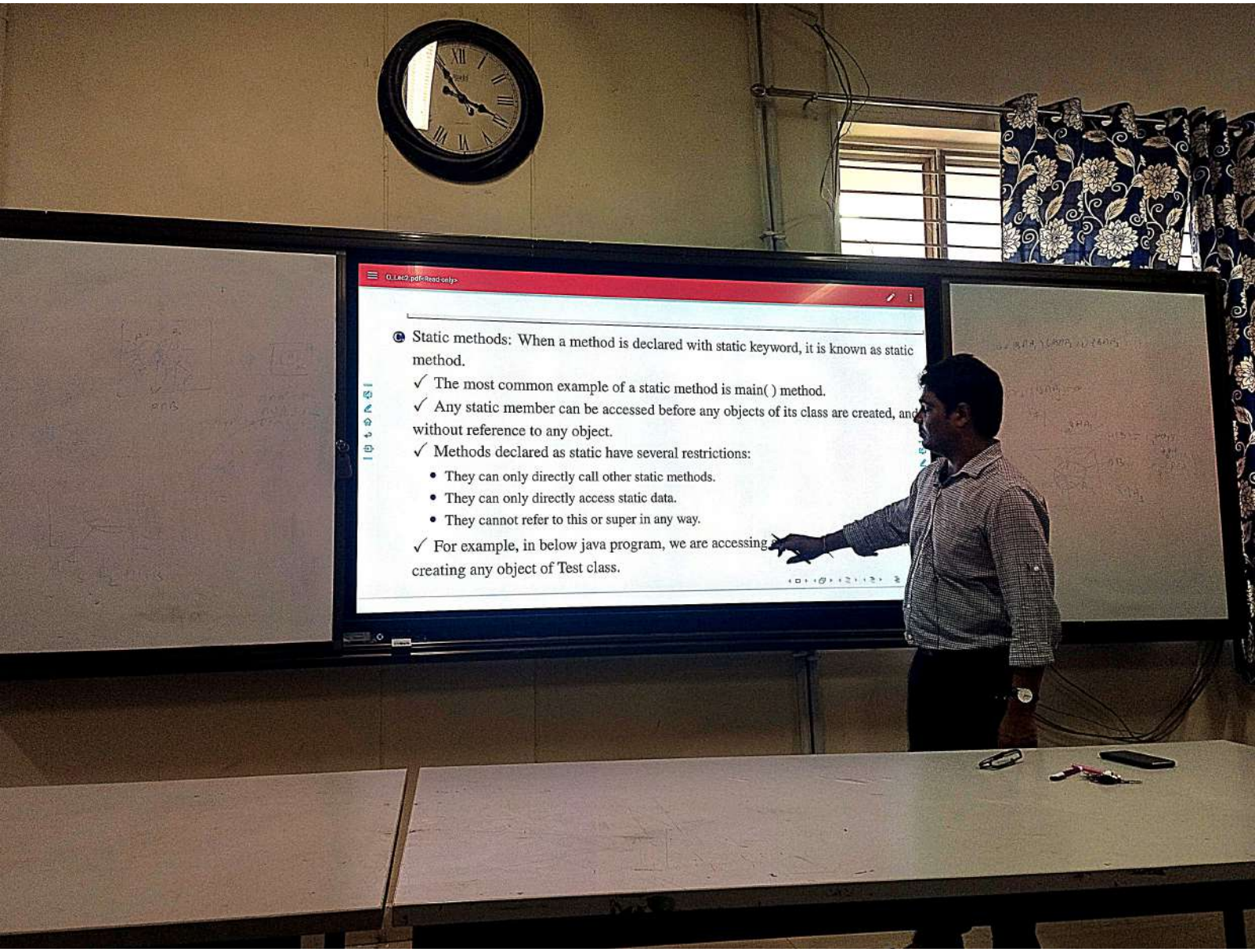

O_Lec2.pdf<Read-only>
Types of Variables XII
// Java program to demonstrate use of static blocks
public class Test
{
// static variable
static int a = 10;
static int b;
// static block
static
{
System.out.println("Static block initialized.");
b = a * 4;
}
public static void main(String[] args)
{
System.out.println("from main");
System.out.println("Value of a : "+a);
System.out.println("Value of b : "+b);
}
}


0_Lec2.pdf<Read-only>
Types of Variables XIII
Static Variables: When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.
Important points for static variables :-
We can create static variables at class-level only. See here
static block and static variables are executed in order they are present in a program.
Types of Variables XIV
100%

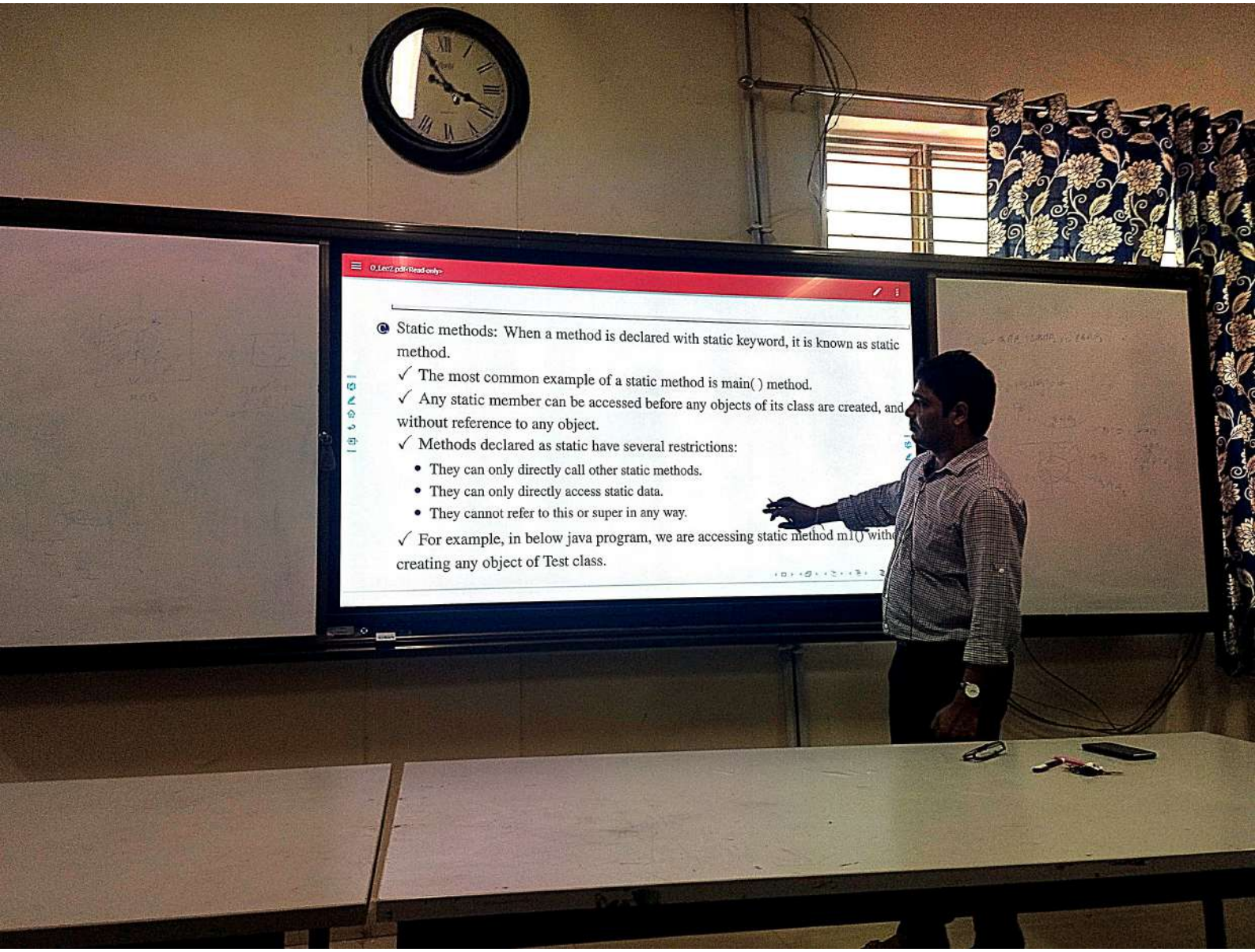

FILE
HOME
REVIEW
SIGN
Static blocks in Java: Java supports a special block, called static block (also called static clause) which can be used for static initializations of a class.
public class Test
{
static int i;
int j;
// start of static block
static
{
i = 10;
System.out.println("static block called ");
}
// end of static block
}
class Main
{
public static void main(String args[])
{
// Although we don't have an object of Test, static block is
// called because i is being accessed in following statement.
System.out.println(Test.i);
}
}
Types of Variables X
✓ Also, static blocks are executed before constructors.
class Test

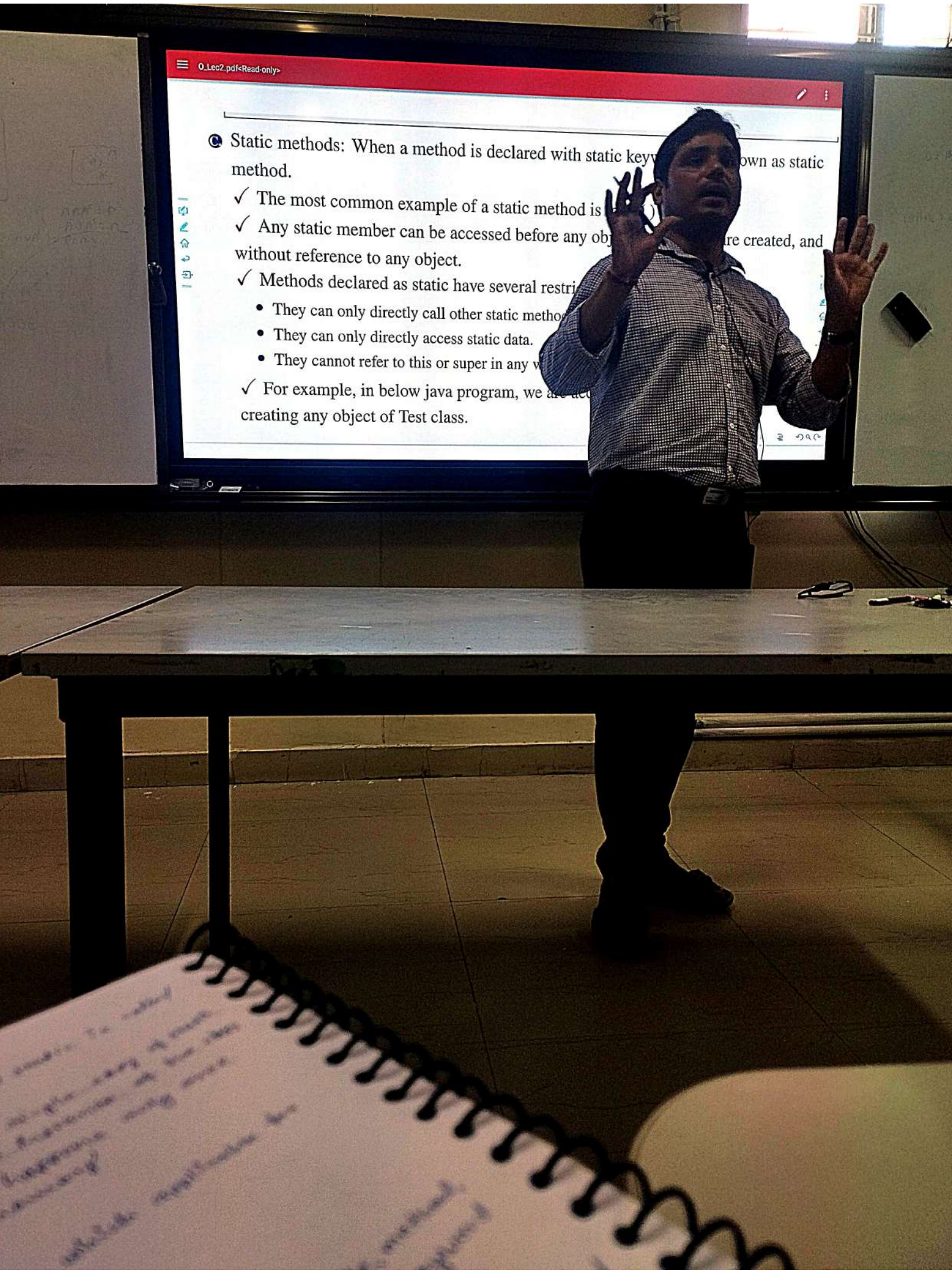

O_Lec2.pdf<Read-only>
Types of Variables XIV
// java program to demonstrate execution of static blocks and variables
public class Test
{
// static variable
static int a = m1();
// static block
static
{
System.out.println("Inside static block");
}
// static method
static int m1()
{
System.out.println("from m1");
return 20;
}
// static method(main !!)
public static void main(String[] args)
{
System.out.println("Value of a : "+a);
System.out.println("from main");
}
}


O_Lec2.pdf<Read-only>
Static methods: When a method is declared with static keyword, it is known as static
ommon example of a static method is main( ) method.
member can be accessed before any objects of its class are created, and
nce to any object.
eclared as static have several restrictions:
directly cal her static methods.
irectly ac static data.
er to thi uper in any way.
va program, we are accessing static method m1() without
st class.


O_Lec2.pdf<Read-only>
Static methods: When a method is declared with static keyword, it is known as static method.
✓ The most common example of a static method is main( ) method.
✓ Any static member can be accessed before any objects of its class are created, and without reference to any object.
✓ Methods declared as static have several restrictions:
• They can only directly call other static methods.
• They can only directly access static data.
• They cannot refer to this or super in any way.
✓ For example, in below java program, we are accessing creating any object of Test class.

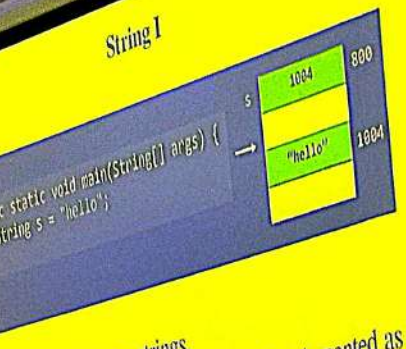

Non-Primitive Data Types or Reference Data Types IX
boolean : false
int : 0
double : 0.0
String : null
User Defined Type : null
import java.io.*;
public class GFG
{
public static void main(String args[]) throws IOException
{
int i, x;
i=10;
x=1;
x=100;
int ar[] =
{
0
1, 2, 3, 4
}
;
int[] d=ar;
d[1]=100;
d =new int[5];


0_Lec2.pdf [Read-only]
Static methods: When a method is declared with static keyword, it is known as static method.
✓ The most common example of a static method is main( ) method.
✓ Any static member can be accessed before any objects of its class are created, and without reference to any object.
✓ Methods declared as static have several restrictions:
• They can only directly call other static methods.
• They can only directly access static data.
• They cannot refer to this or super in any way.
✓ For example, in below java program, we are accessing static method m1() witho
creating any object of Test class.


0_Lec2.pdf<Read-only>
Static methods: When a method is declared with static key
own as static
method.
✓ The most common example of a static method is
✓ Any static member can be accessed before any ob
re created, and
without reference to any object.
✓ Methods declared as static have several restri
They can only directly call other metho
They can only directly access static data.
They cannot refer to this or super in any w
✓ For example, in below java program, we
creating any object of Test class.

String I

```
public static void main(String[] args) {
    String s = "hello";
}
```



✓ The String class represents character strings.

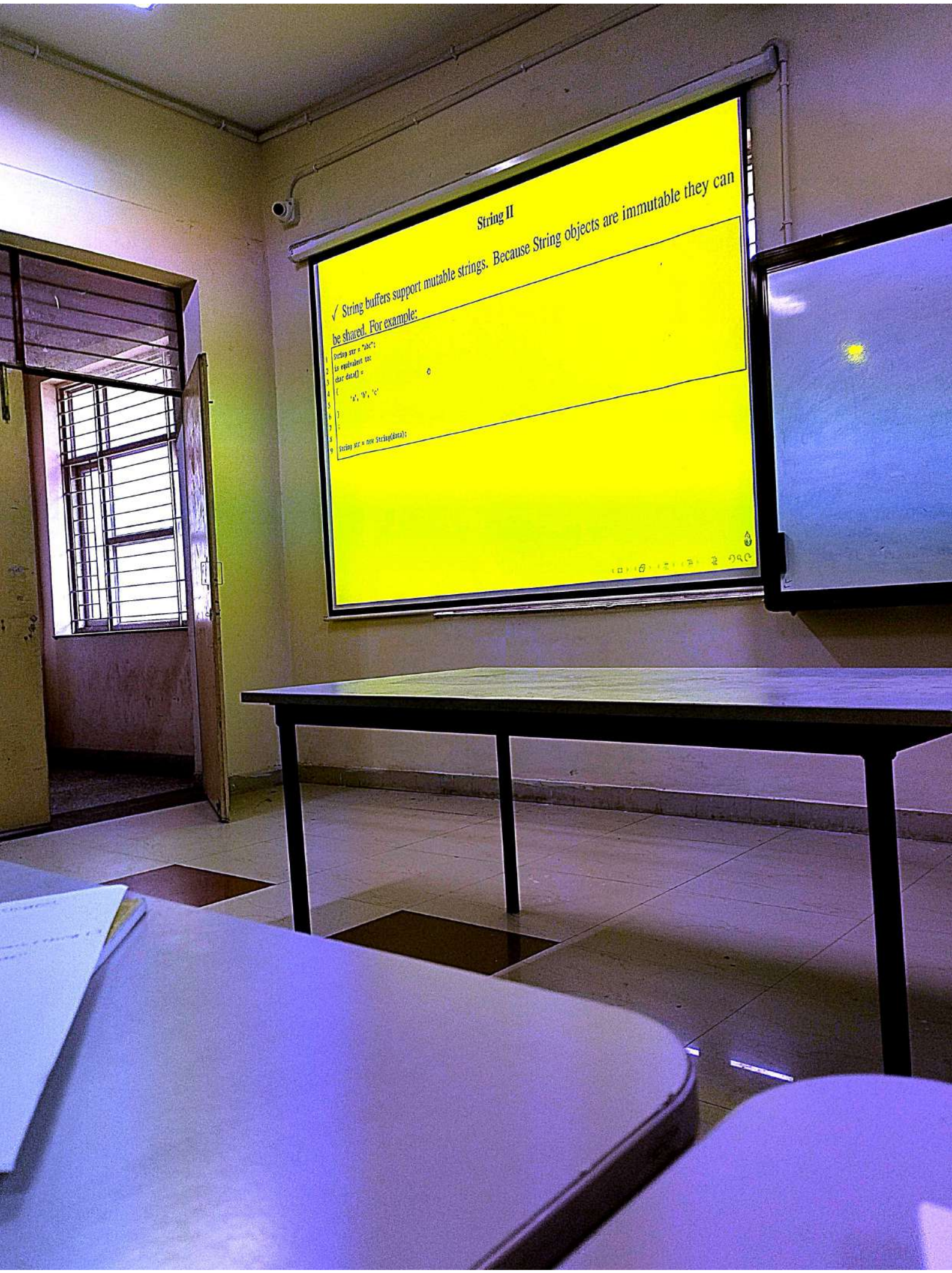
✓ All string literals in Java programs, such as "abc", are implemented as instances of this class.

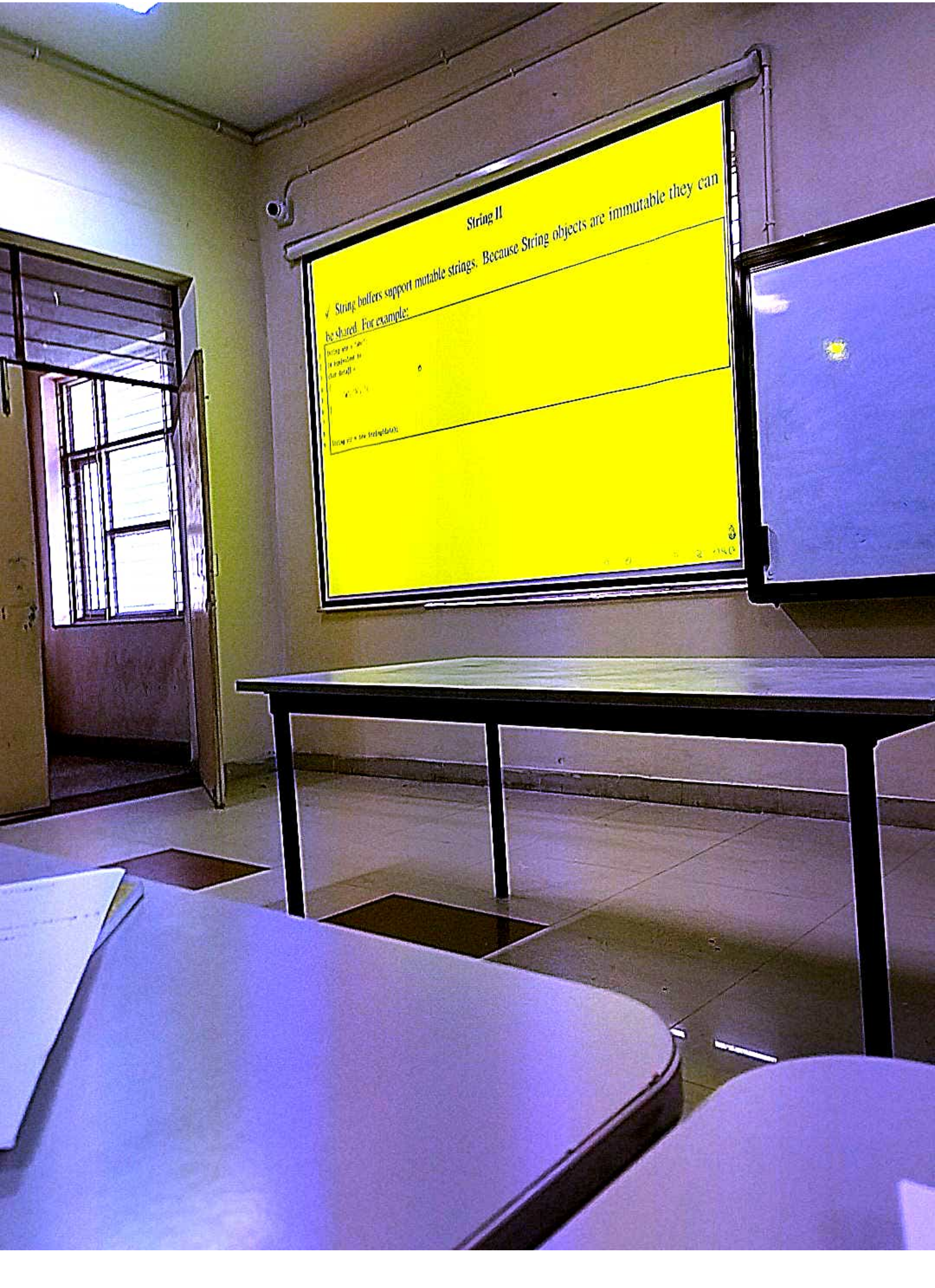
✓ Strings in Java are Objects that are backed internally by a char array.

✓ Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

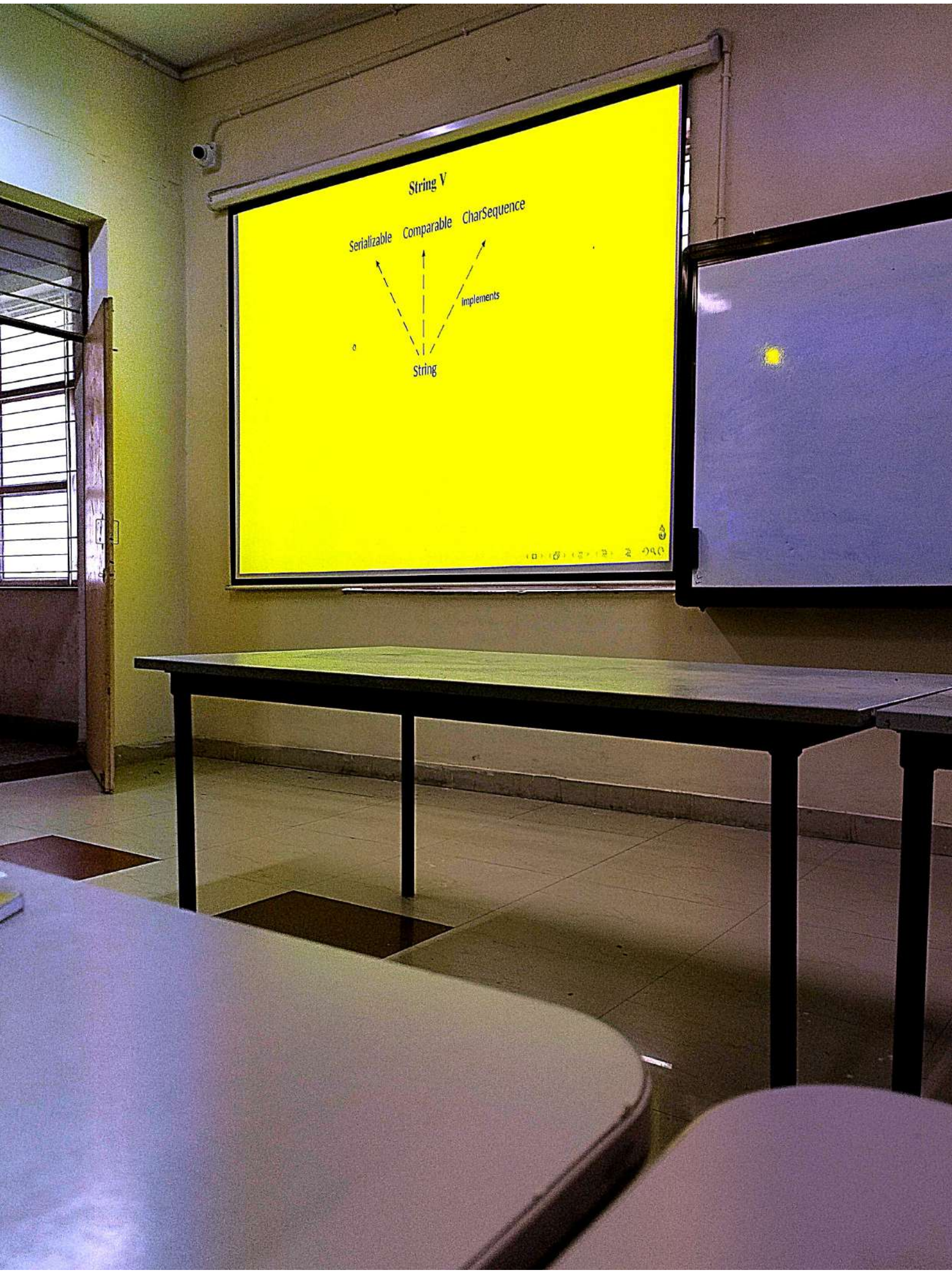
✓ Strings are constant; their values cannot be changed after they are created.


String III
Here are some more examples of how strings can be used:
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2, 3);
String d = cde.substring(1, 2);
char[] ch:
{
'a','b','c', 'd', 'e','f','g', 'h', 'i', 'j', 'k', 'l'
}
String s=new String(ch);
or
String s="javatpoint";


String II
✓ String buffers support mutable strings. Because String objects are immutable they can be shared. For example:
String str = "abc";
is equivalent to:
char data[] = {
'a', 'b', 'c'
};
String str = new String(data);


String II
String buffers support mutable strings. Because String objects are immutable they can
be shared. For example:

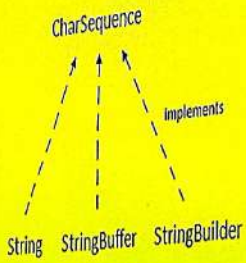

String IV
✓ Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
✓ The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.

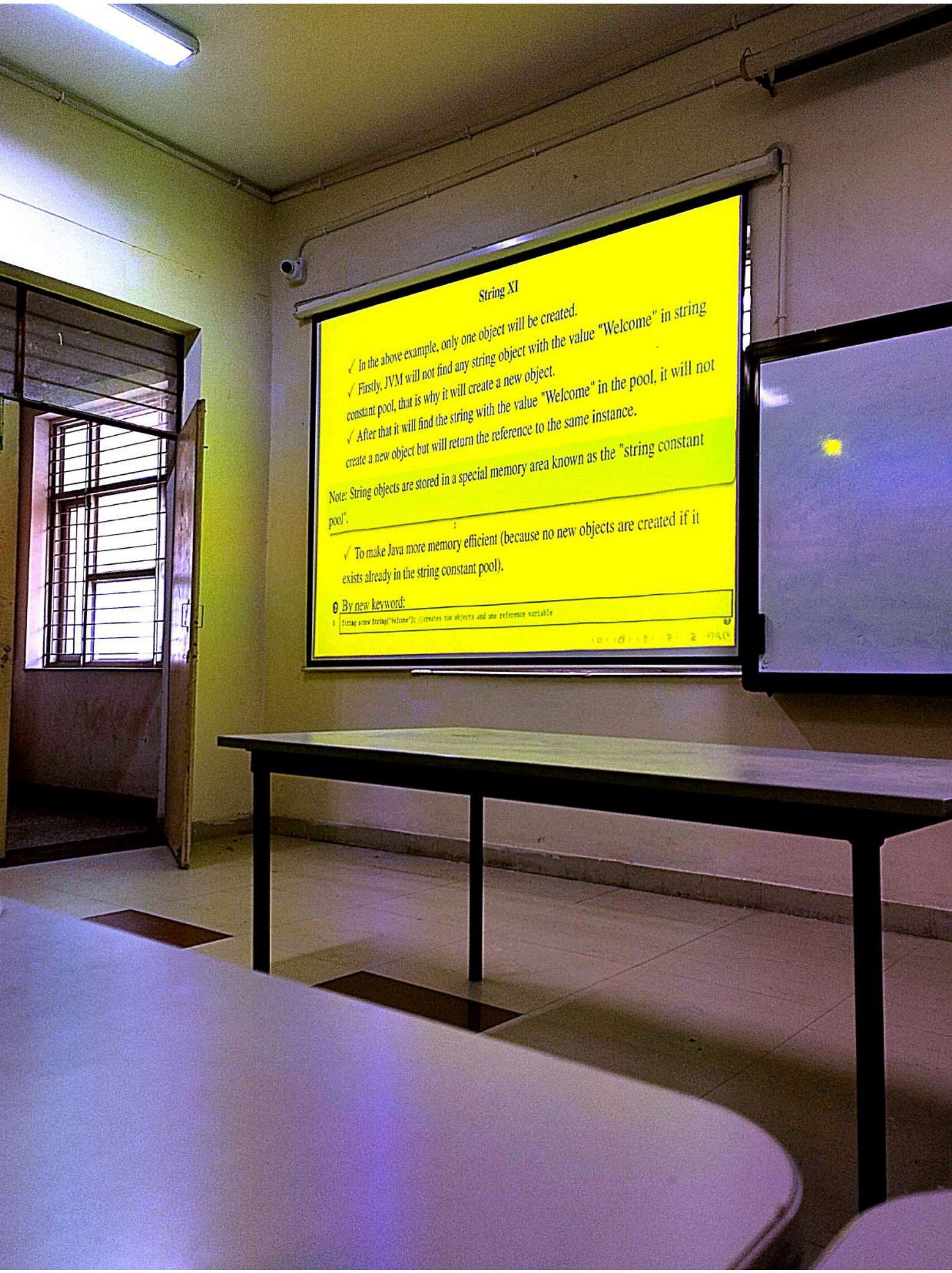

String V
Serializable
Comparable
CharSequence
implements
String

String VI

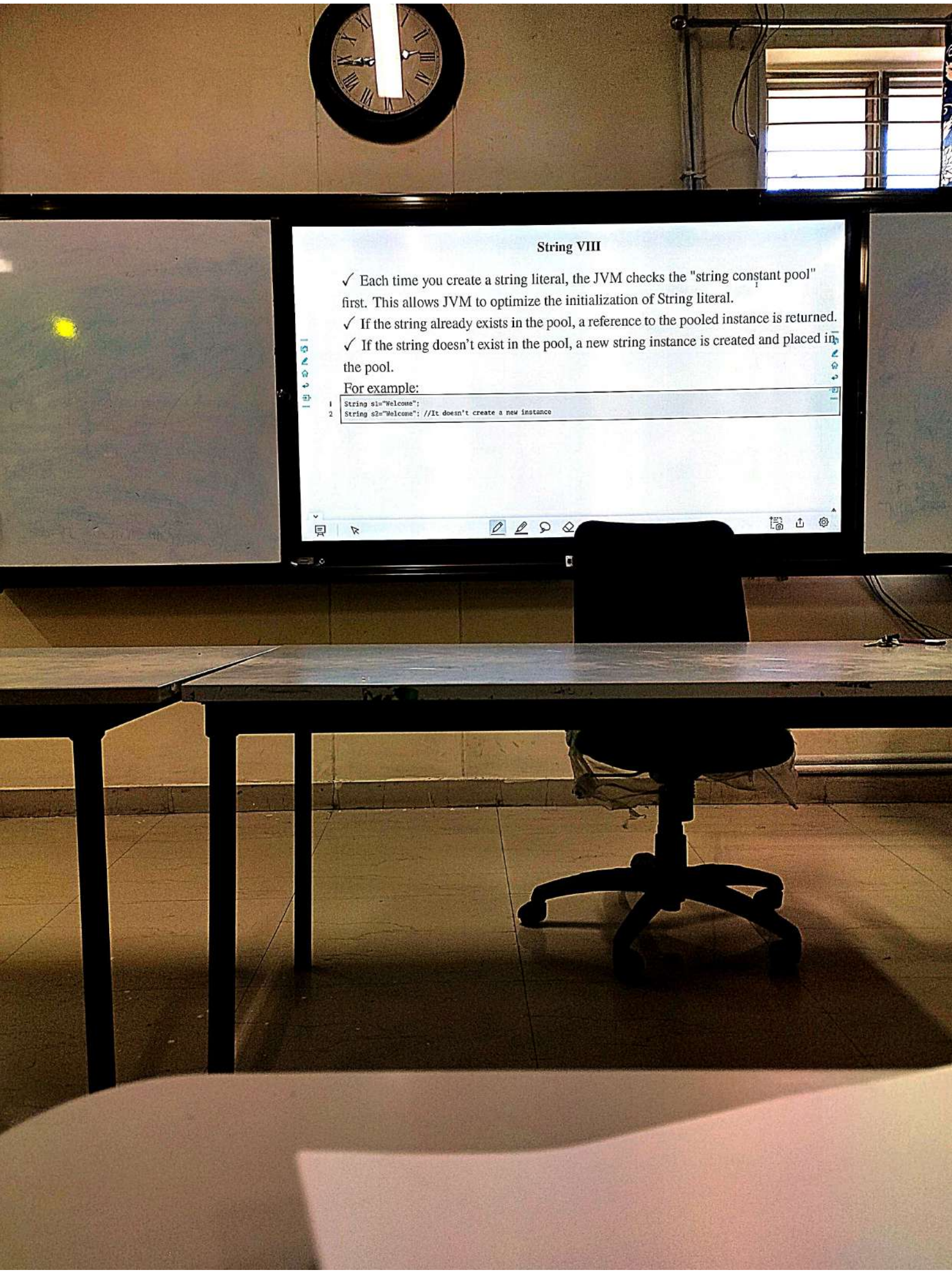
CharSequence Interface:

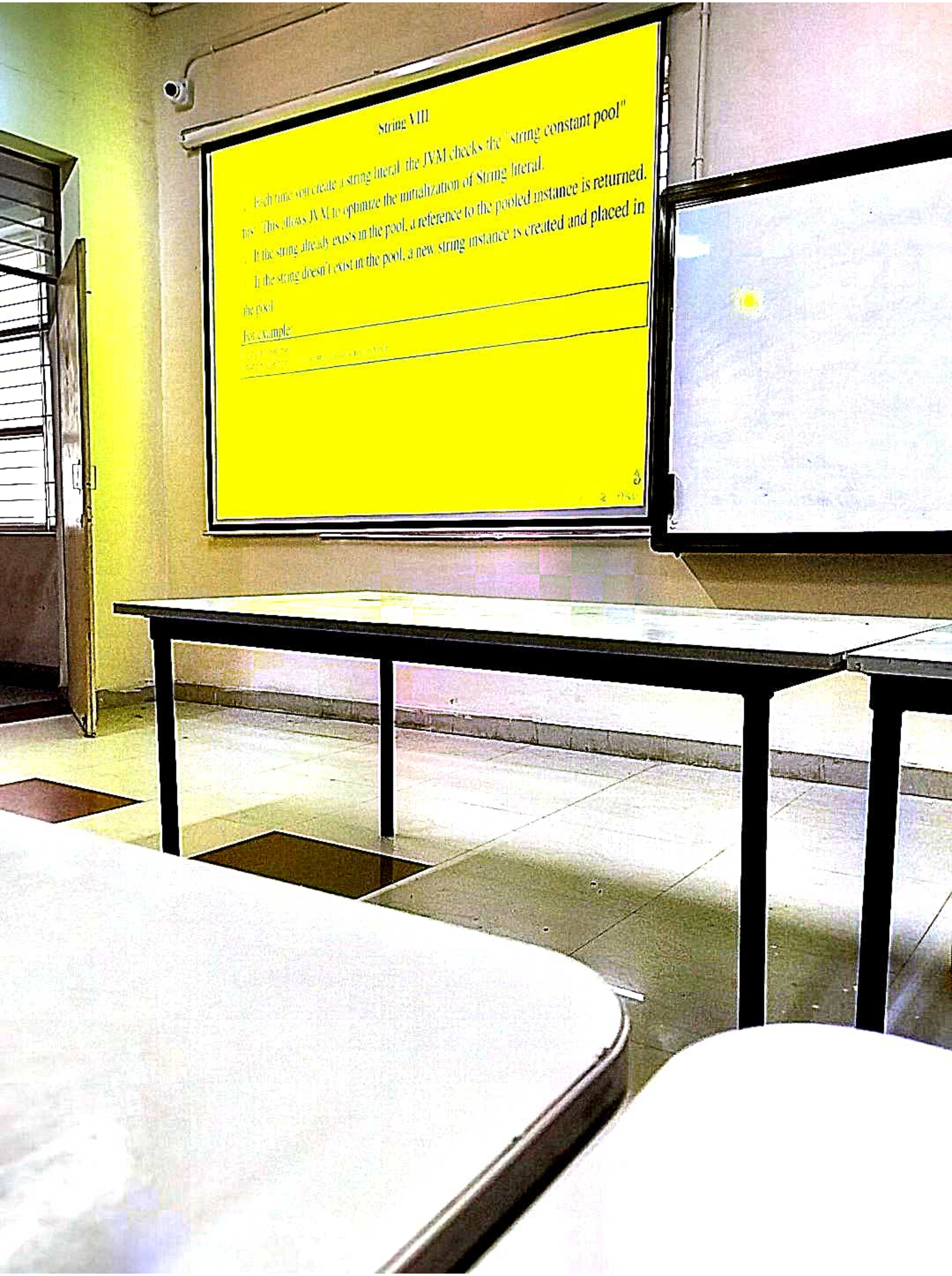
✓ The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.

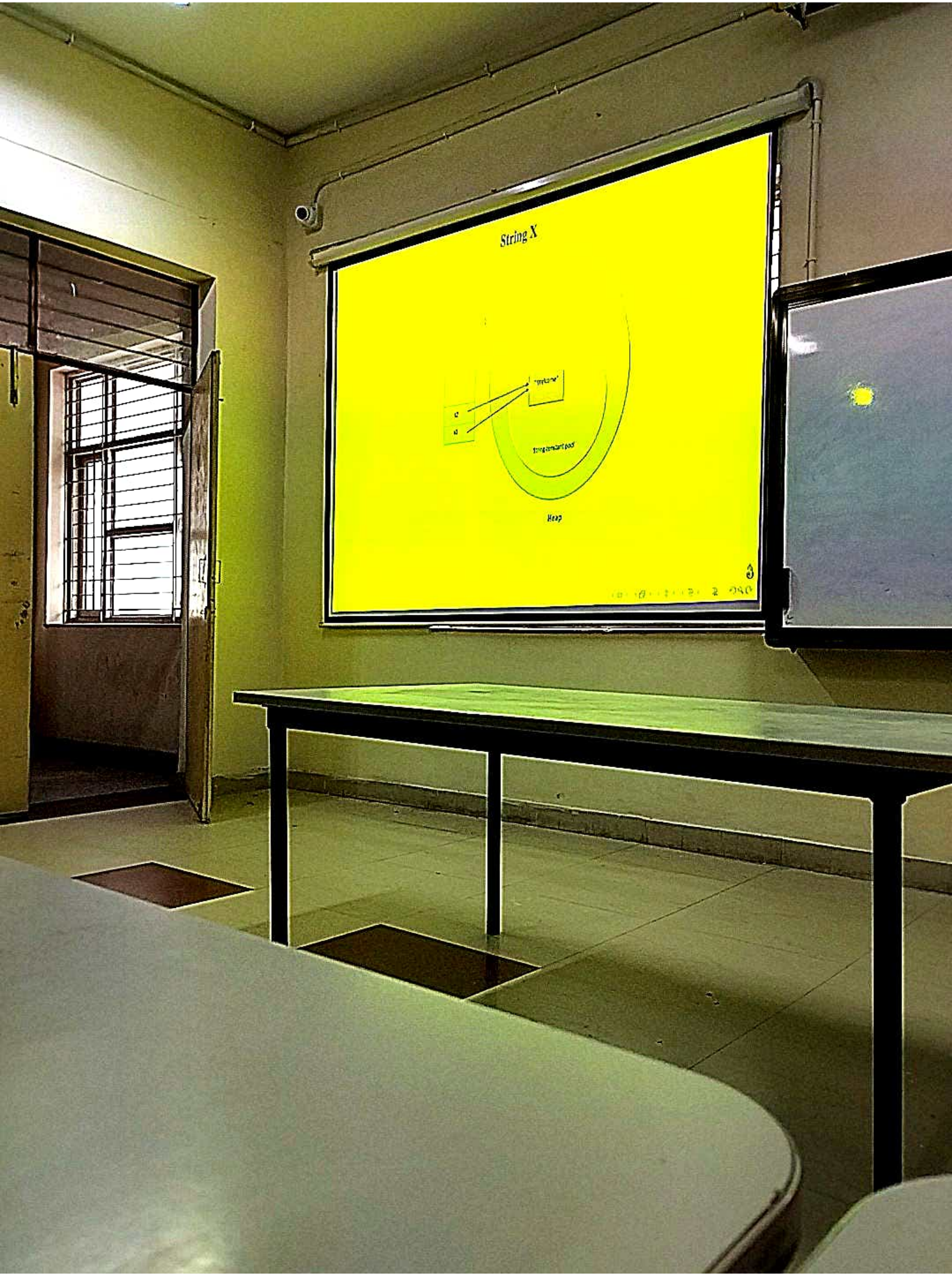


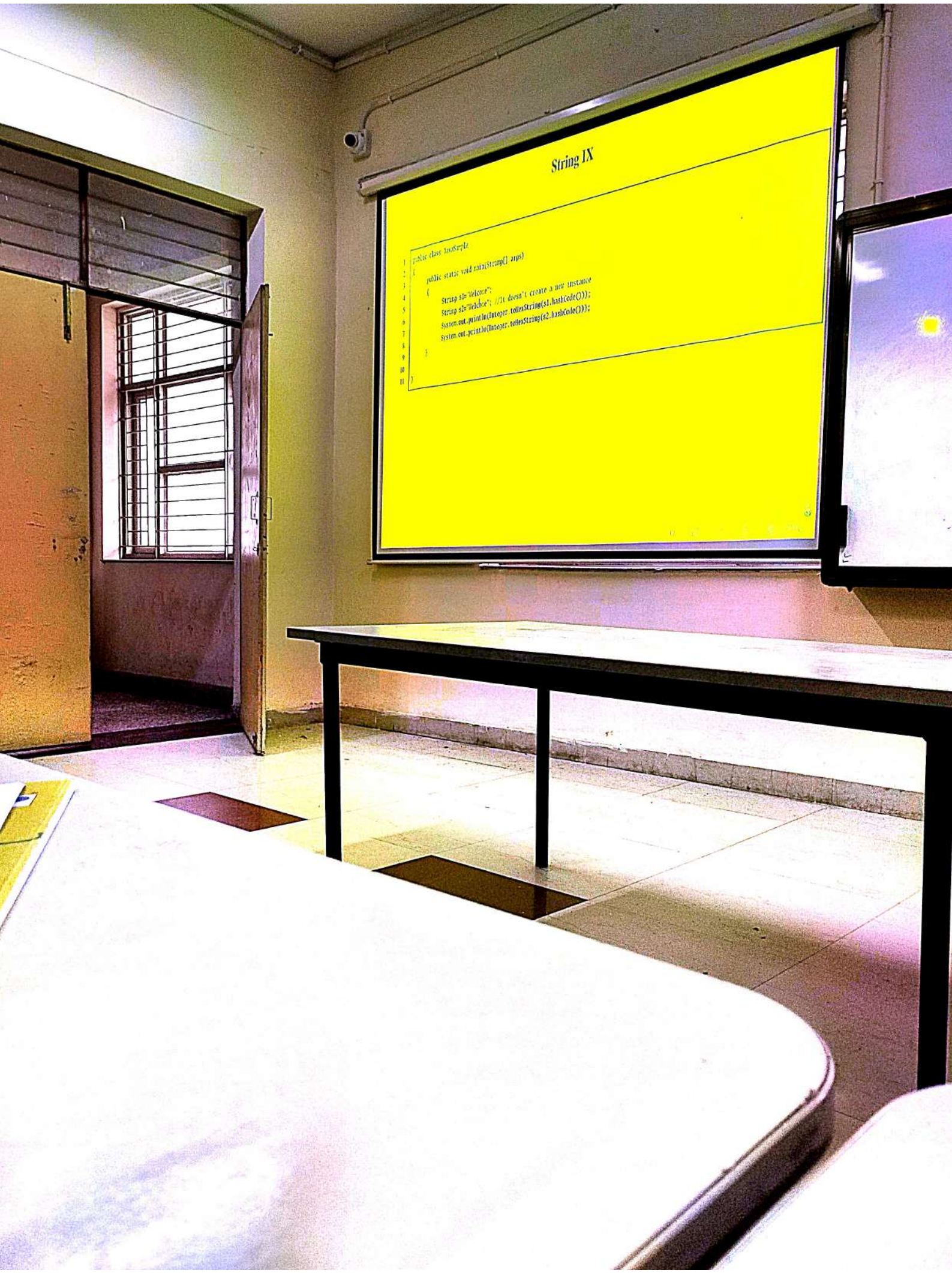

String XI
✓ In the above example, only one object will be created.
✓ Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object.
✓ After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.
Note: String objects are stored in a special memory area known as the "string constant pool".
✓ To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).
By new keyword:
String s=new String("Welcome"); //creates two objects and one reference variable

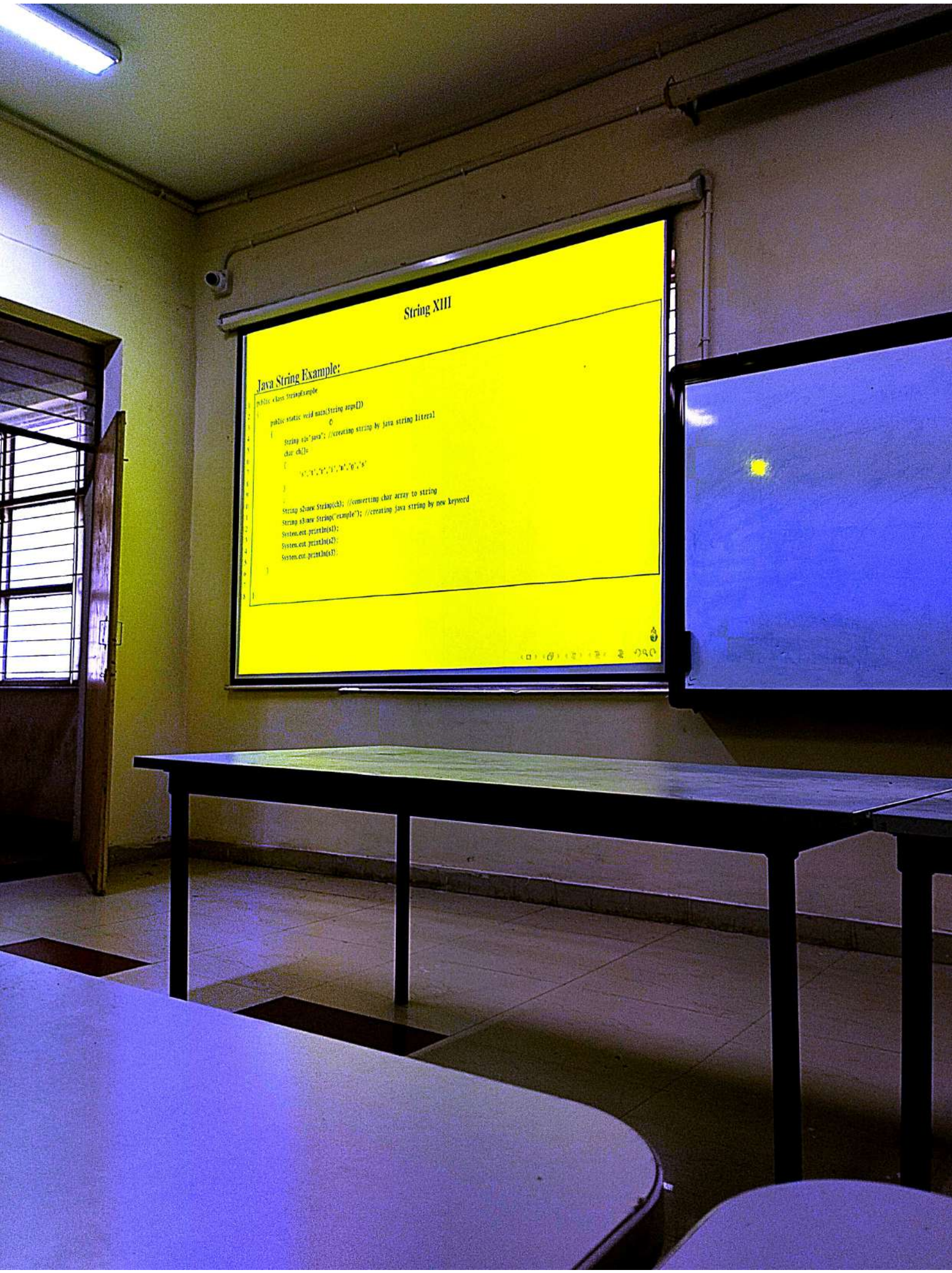

String VII
✓ The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.
How to create a string object?
There are two ways to create String object:
1 By string literal
2 By new keyword
1 String Literal : Java String literal is created by using double quotes. For Example:
1 String s="welcome";

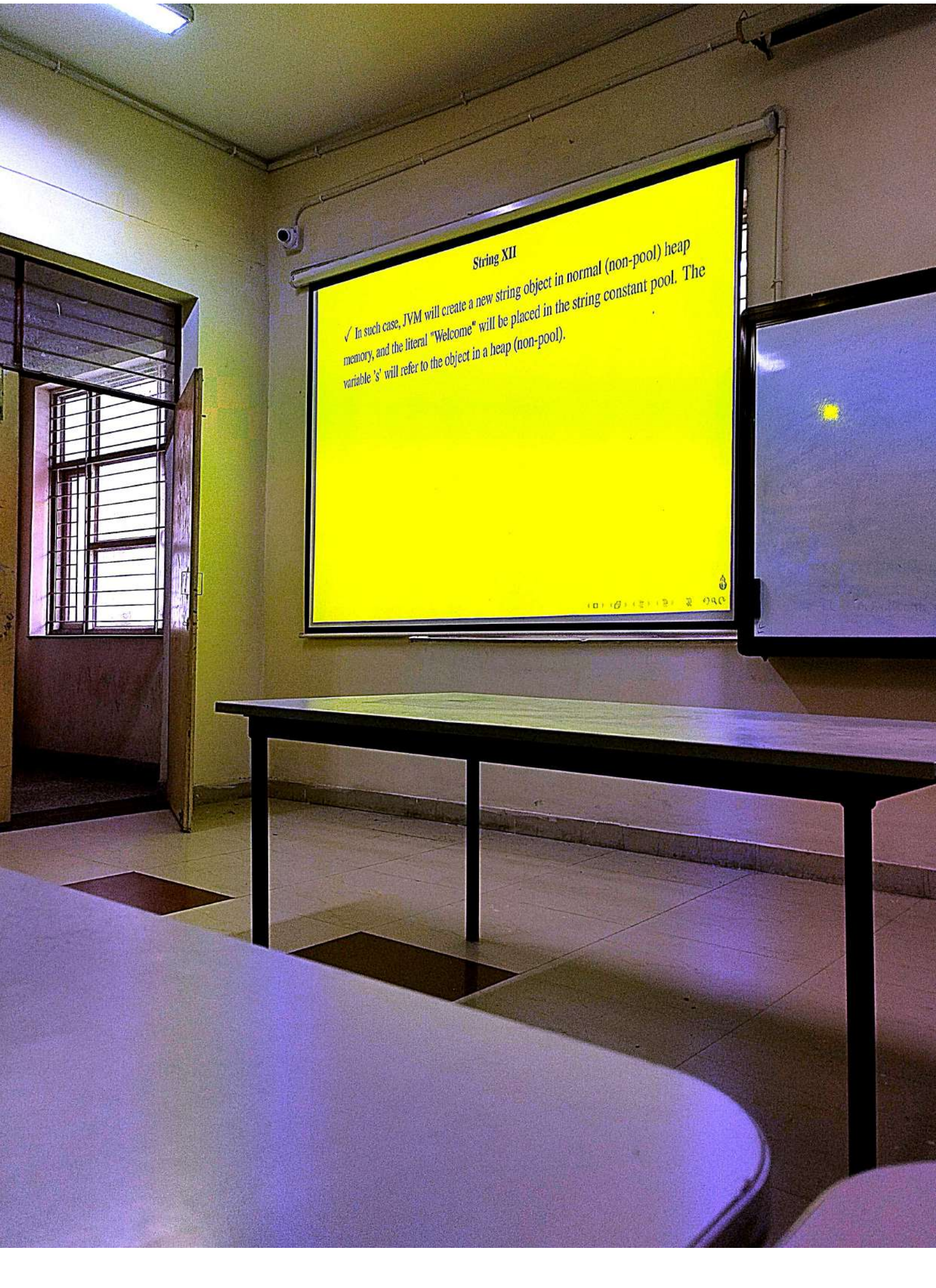

String VIII
✓ Each time you create a string literal, the JVM checks the "string constant pool" first. This allows JVM to optimize the initialization of String literal.
✓ If the string already exists in the pool, a reference to the pooled instance is returned.
✓ If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
For example:
1 String s1="Welcome";
2 String s2="Welcome"; //It doesn't create a new instance

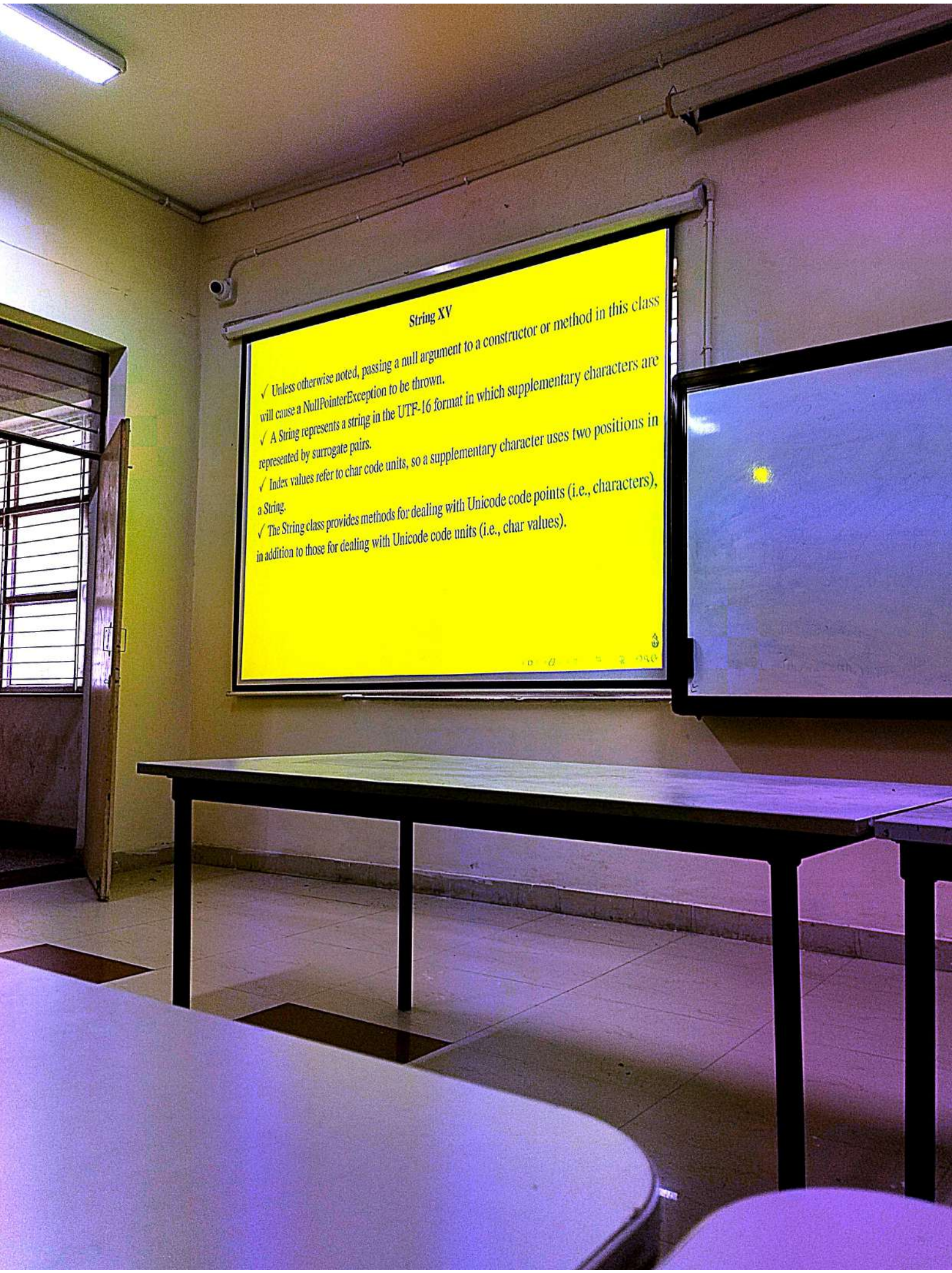

String VIII
Each time you create a string literal the JVM checks the "string constant pool"
This allows JVM to optimize the initialization of String literal.
If the string already exists in the pool, a reference to the pooled instance is returned.
If the string doesn't exist in the pool, a new string instance is created and placed in
the pool
For example

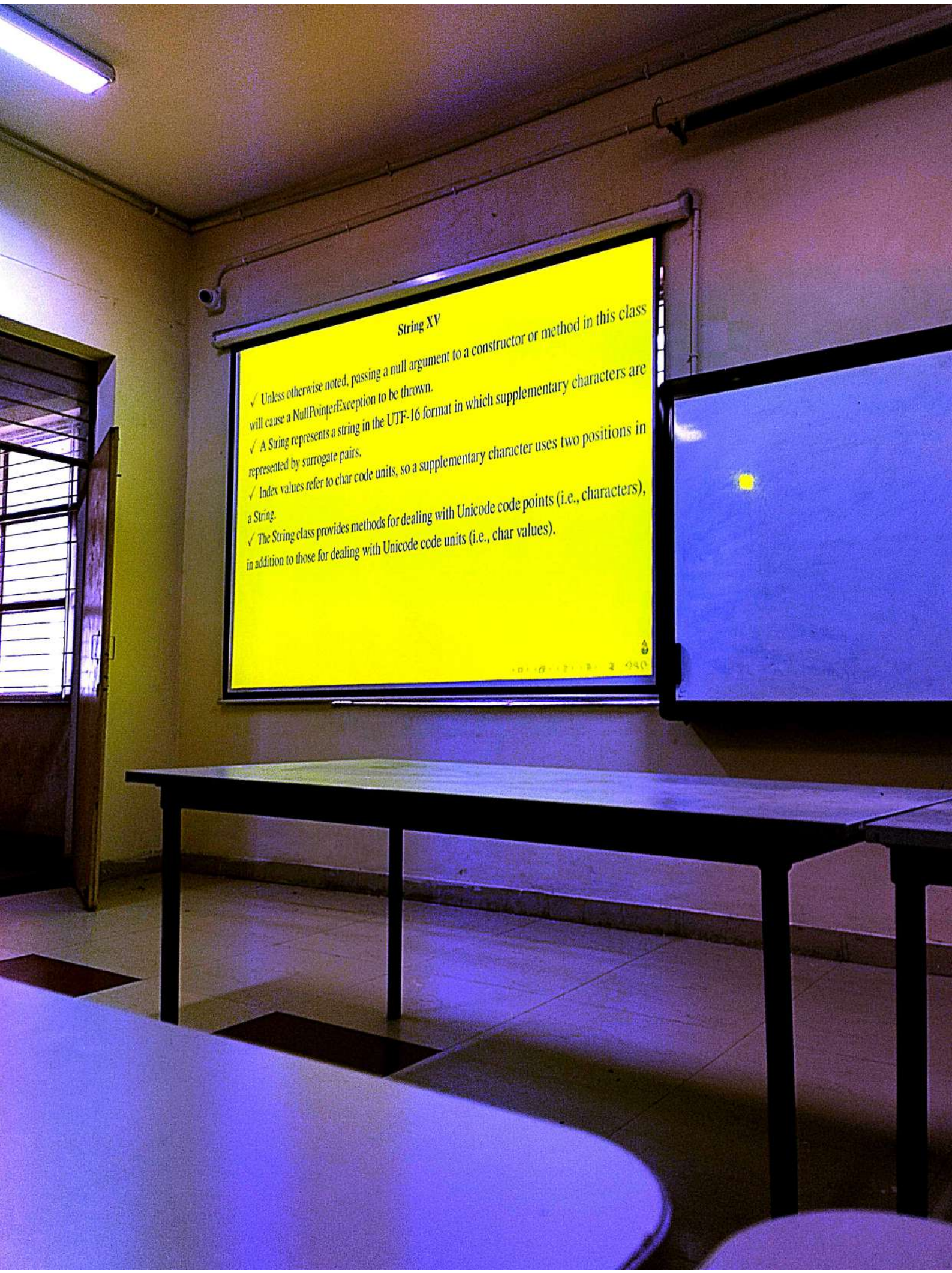

String X
"immutable"
x2
x3
String constant pool
Heap


String IX
public class StrStyle
{
public static void main(String[] args)
{
String s1="Welcome";
String s2="Welcome"; //It doesn't create a new instance
System.out.println(Integer.toHexString(s1.hashCode()));
System.out.println(Integer.toHexString(s2.hashCode()));
}
}

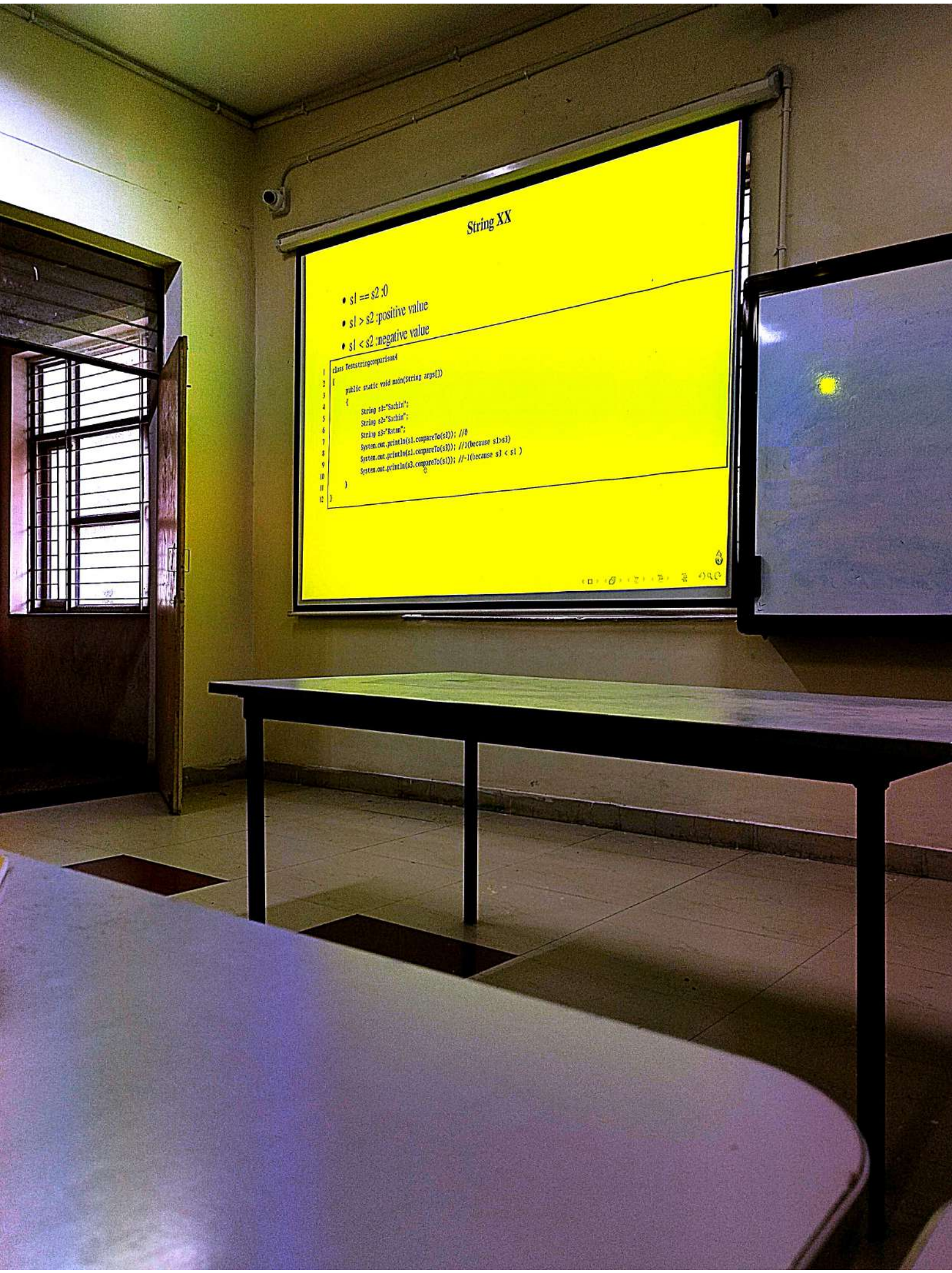

String XIII
Java String Example:
public class StringExample
{
public static void main(String args[])
{
String s1="java"; //creating string by java string literal
char ch[]={
'j','a','v','a','p','o','s'
};
String s2=new String(ch); //converting char array to string
String s3=new String("example"); //creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}
}

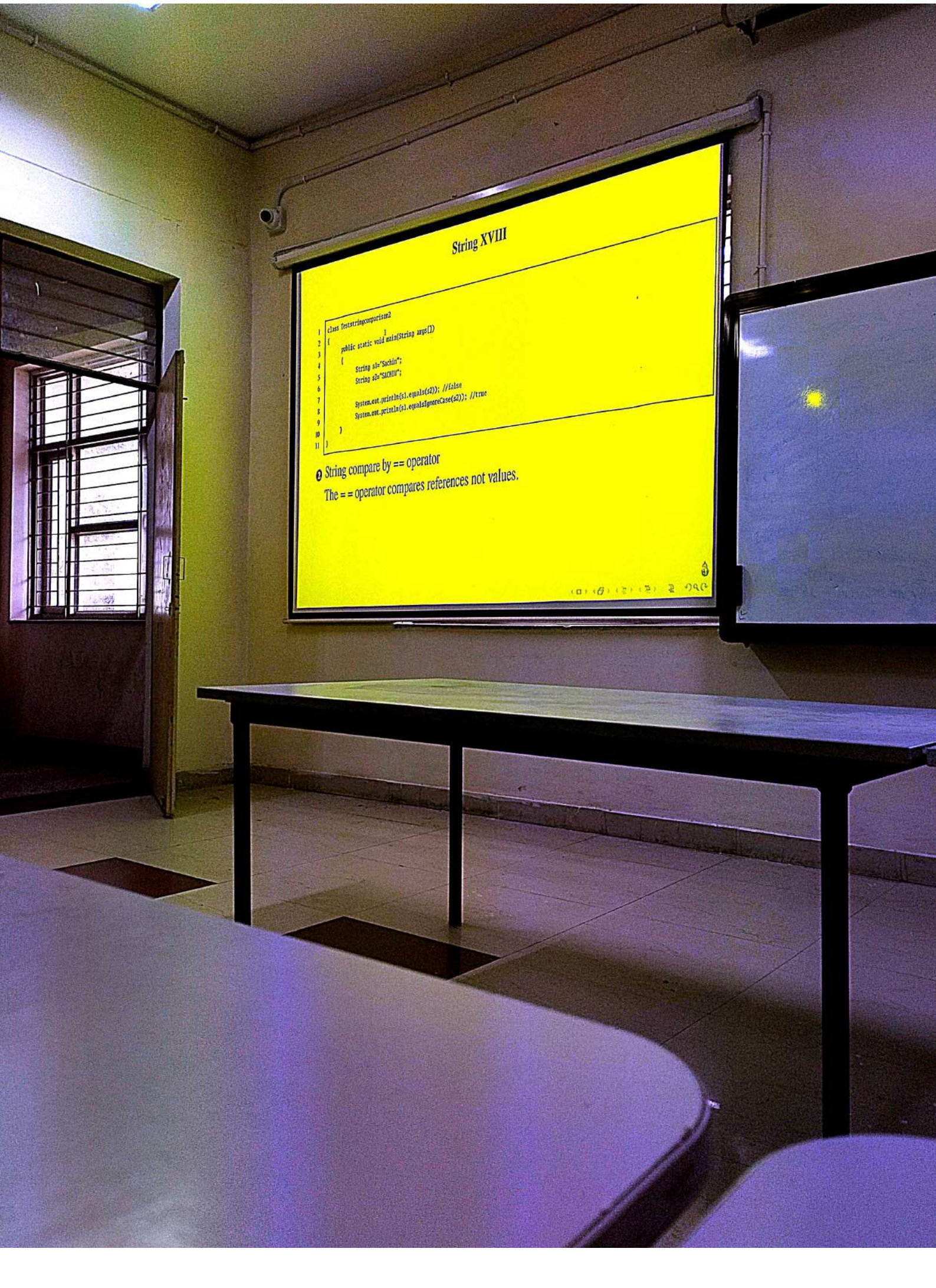

String XII
✓ In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable 's' will refer to the object in a heap (non-pool).

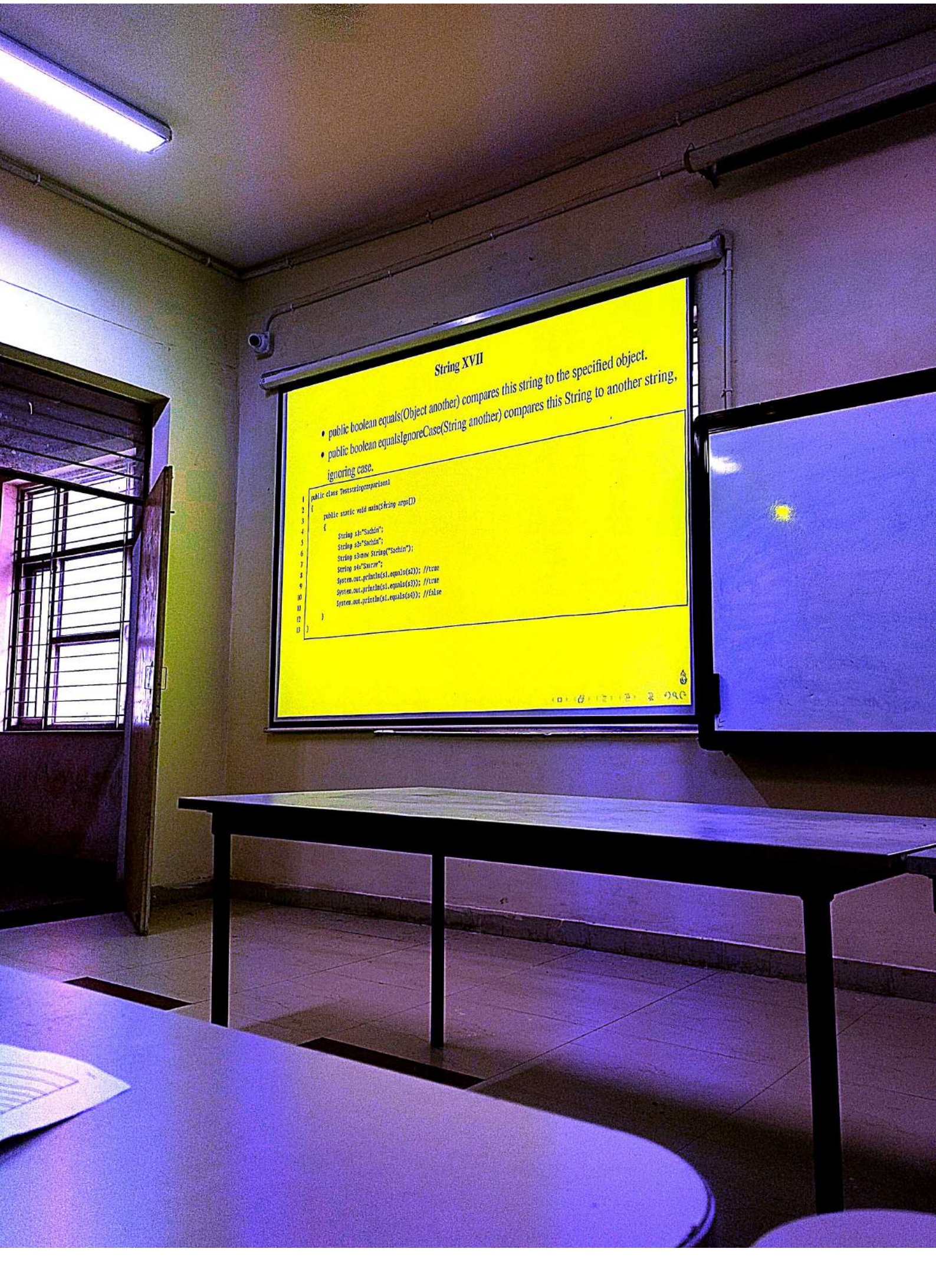

String XV
✓ Unless otherwise noted, passing a null argument to a constructor or method in this class will cause a NullPointerException to be thrown.
✓ A String represents a string in the UTF-16 format in which supplementary characters are represented by surrogate pairs.
✓ Index values refer to char code units, so a supplementary character uses two positions in a String.
✓ The String class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., char values).

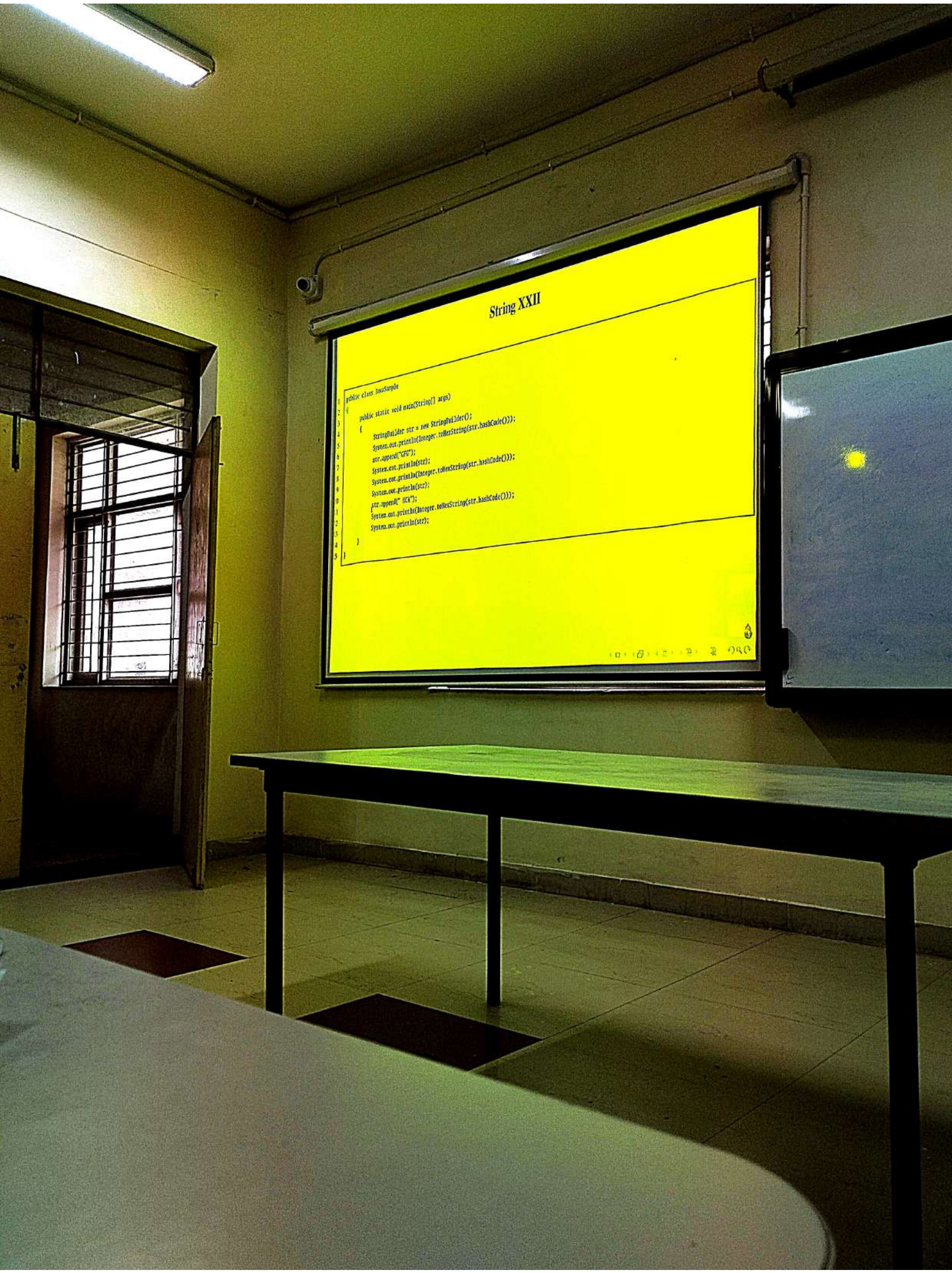

String XV
✓ Unless otherwise noted, passing a null argument to a constructor or method in this class will cause a NullPointerException to be thrown.
✓ A String represents a string in the UTF-16 format in which supplementary characters are represented by surrogate pairs.
✓ Index values refer to char code units, so a supplementary character uses two positions in a String.
✓ The String class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., char values).

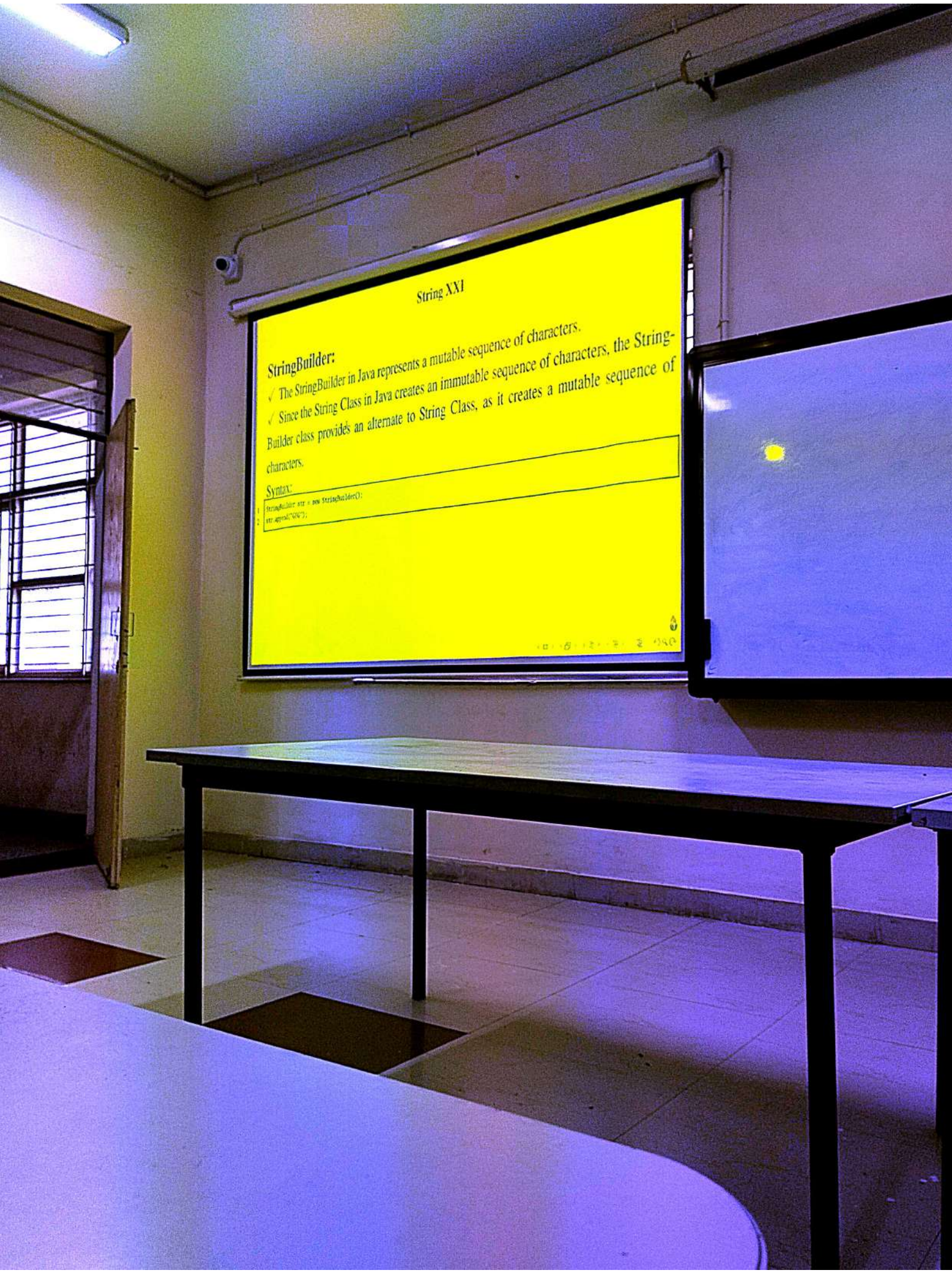

String XVI
Java String compare:
✓ We can compare string in java on the basis of content and reference.
✓ It is used in authentication (by equals() method), sorting (by compareTo() method), reference matching (by == operator) etc.
✓ There are three ways to compare string in java:
By equals() method
By = = operator
By compareTo() method
String compare by equals() method
The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

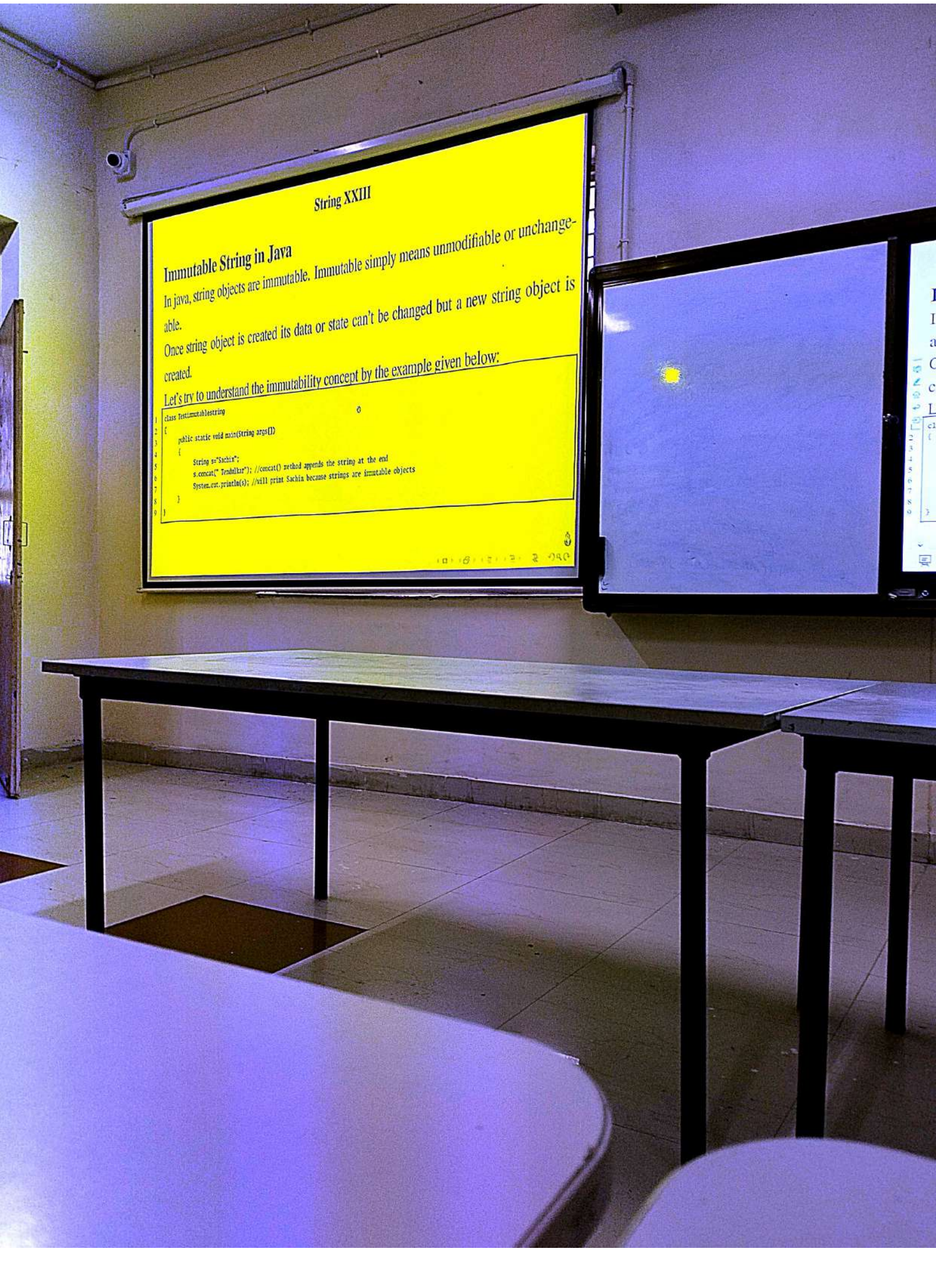

String XX
• s1 == s2 :0
• s1 > s2 :positive value
• s1 < s2 :negative value
class Teststringcomparison4
{
public static void main(String args[])
{
String s1="Sachin";
String s2="Sachin";
String s3="Ratan";
System.out.println(s1.compareTo(s2)); //0
System.out.println(s1.compareTo(s3)); //1(because s1>s3)
System.out.println(s3.compareTo(s1)); //-1(because s3 < s1)
}
}

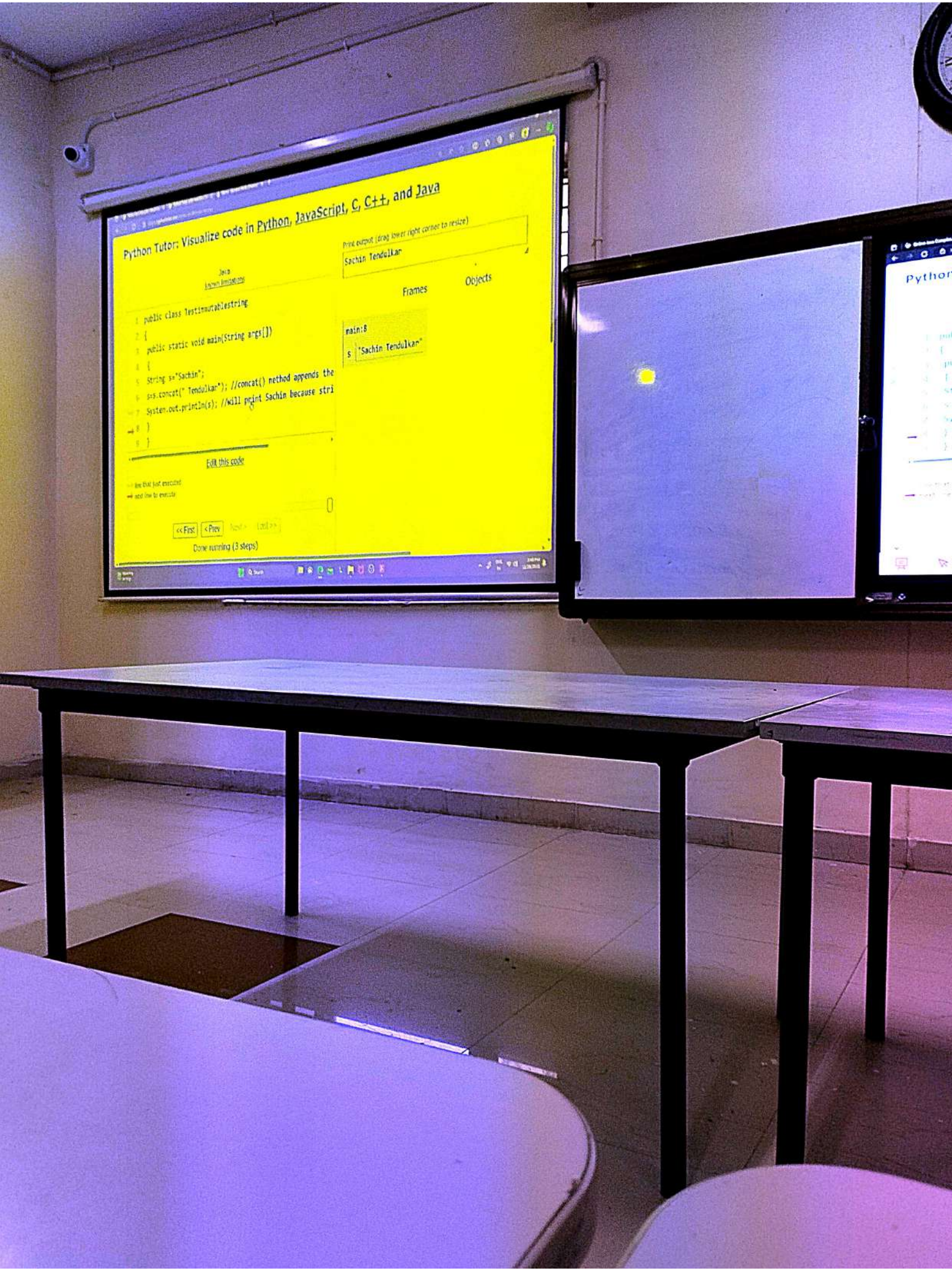

String XVIII
class Teststringcomparison2
{
public static void main(String args[])
{
String s1="Sachin";
String s2="SACHIN";
System.out.println(s1.equals(s2)); //false
System.out.println(s1.equalsIgnoreCase(s2)); //true
}
}
String compare by == operator
The = = operator compares references not values.


String XVII
• public boolean equals(Object another) compares this string to the specified object.
• public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.
public class Teststringcomparison1
{
public static void main(String args[])
{
String s1="Sachin";
String s2="Sachin";
String s3=new String("Sachin");
String s4="Saurav";
System.out.println(s1.equals(s2)); //true
System.out.println(s1.equals(s3)); //true
System.out.println(s1.equals(s4)); //false
}
}

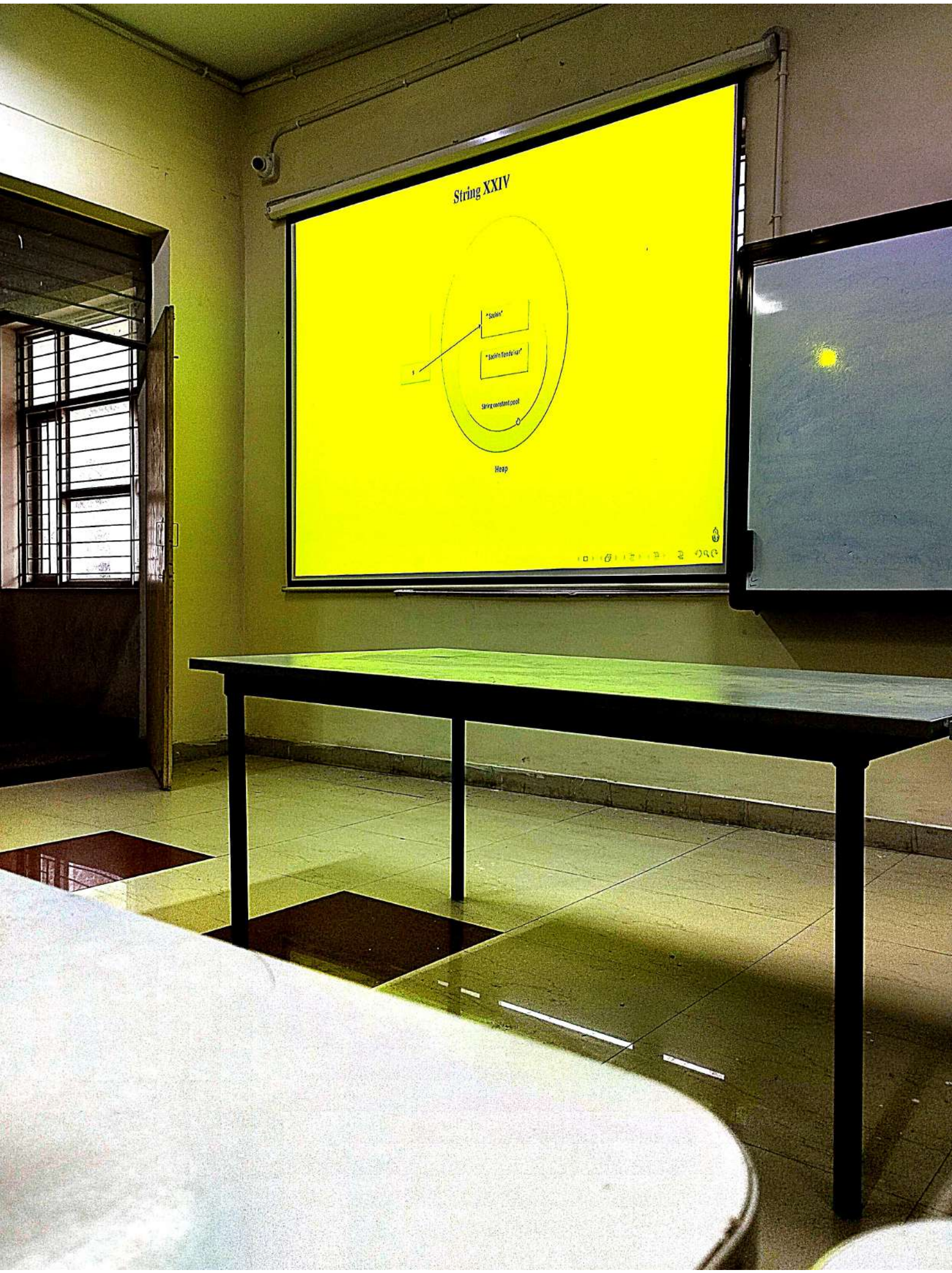

String XXII
public class JavaSample
{
public static void main(String[] args)
{
StringBuilder str = new StringBuilder();
System.out.println(Integer.toHexString(str.hashCode()));
str.append("GFG");
System.out.println(str);
System.out.println(Integer.toHexString(str.hashCode()));
System.out.println(str);
str.append(" HCA");
System.out.println(Integer.toHexString(str.hashCode()));
System.out.println(str);
}
}


String XXI
StringBuilder:
✓ The StringBuilder in Java represents a mutable sequence of characters.
✓ Since the String Class in Java creates an immutable sequence of characters, the String-Builder class provides an alternate to String Class, as it creates a mutable sequence of characters.
Syntax:
1 StringBuilder str = new StringBuilder();
2 str.append("GFG");


String XXIII
Immutable String in Java
In java, string objects are immutable. Immutable simply means unmodifiable or unchange-
able.
Once string object is created its data or state can't be changed but a new string object is
created.
Let's try to understand the immutability concept by the example given below:
class Testimmutablestring
{
public static void main(String args[])
{
String s="Sachin";
s.concat(" Tendulkar"); //concat() method appends the string at the end
System.out.println(s); //will print Sachin because strings are immutable objects
}
}


Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java
Java
known limitations
1 public class TestImmutableString
2 {
3 public static void main(String args[])
4 {
5 String s="Sachin";
6 s.concat(" Tendulkar"); //concat() method appends the
7 System.out.println(s); //will print Sachin because stri
8 }
9 }
Edit this code
line that just executed
next line to execute
<< First
< Prev
Done running (3 steps)
Print output (drag lower right corner to resize)
Sachin Tendulkar
Frames
Objects
main:8
s
"Sachin Tendulkar"


String XXVI
String Concatenation in Java:
✓ In java, string concatenation forms a new string that is the combination of multiple strings. There are two ways to concat string in java:
1 By + (string concatenation) operator
2 By concat() method
1 By + (string concatenation) operator
class TestStringConcatenation1
{
public static void main(String args[])
{
String s="Sachin"+" Tendulkar";
System.out.println(s); //Sachin Tendulkar
}
}

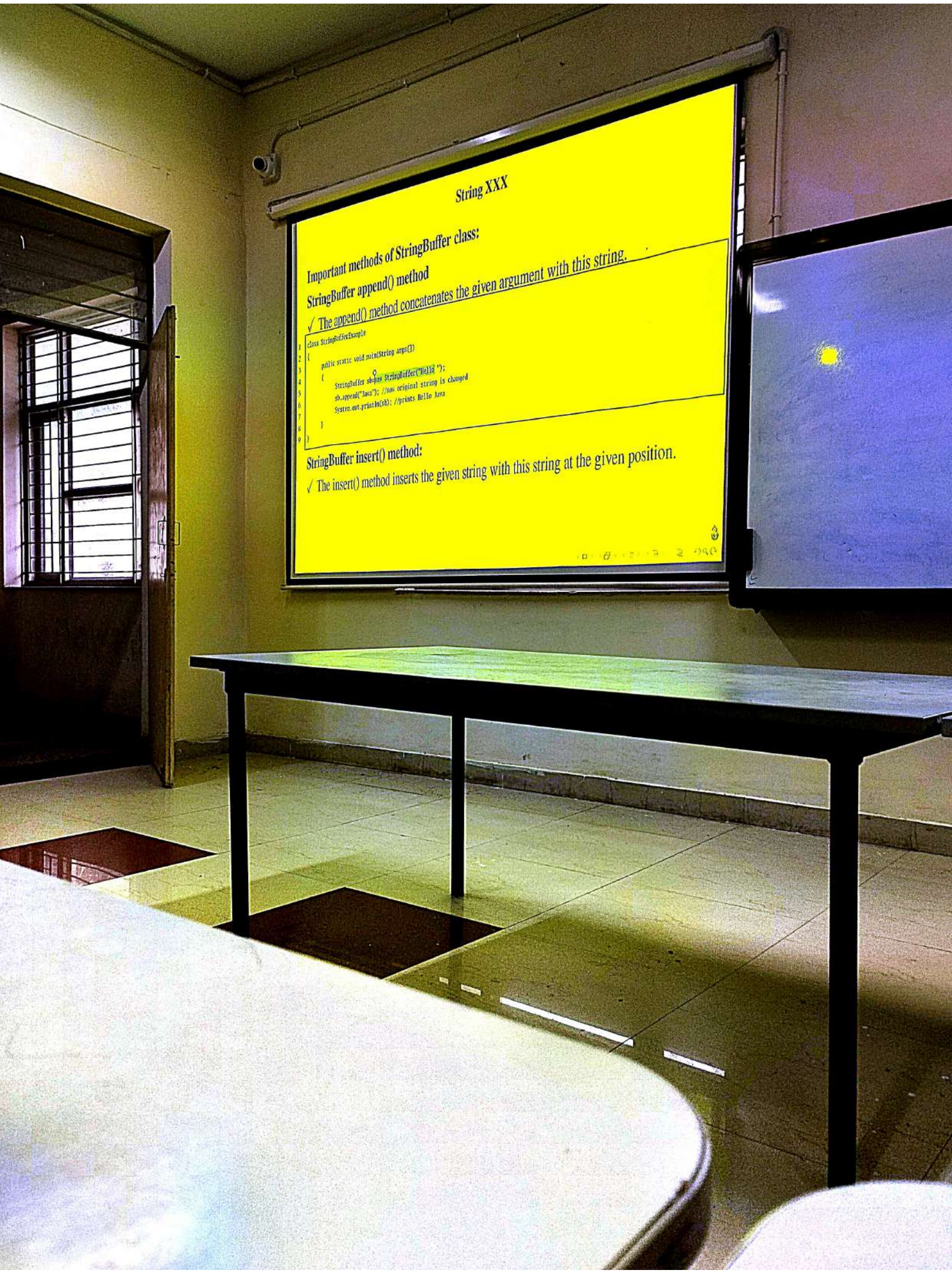

String XXIV
"Sachin"
"Sachin Tendulkar"
String constant pool
Heap

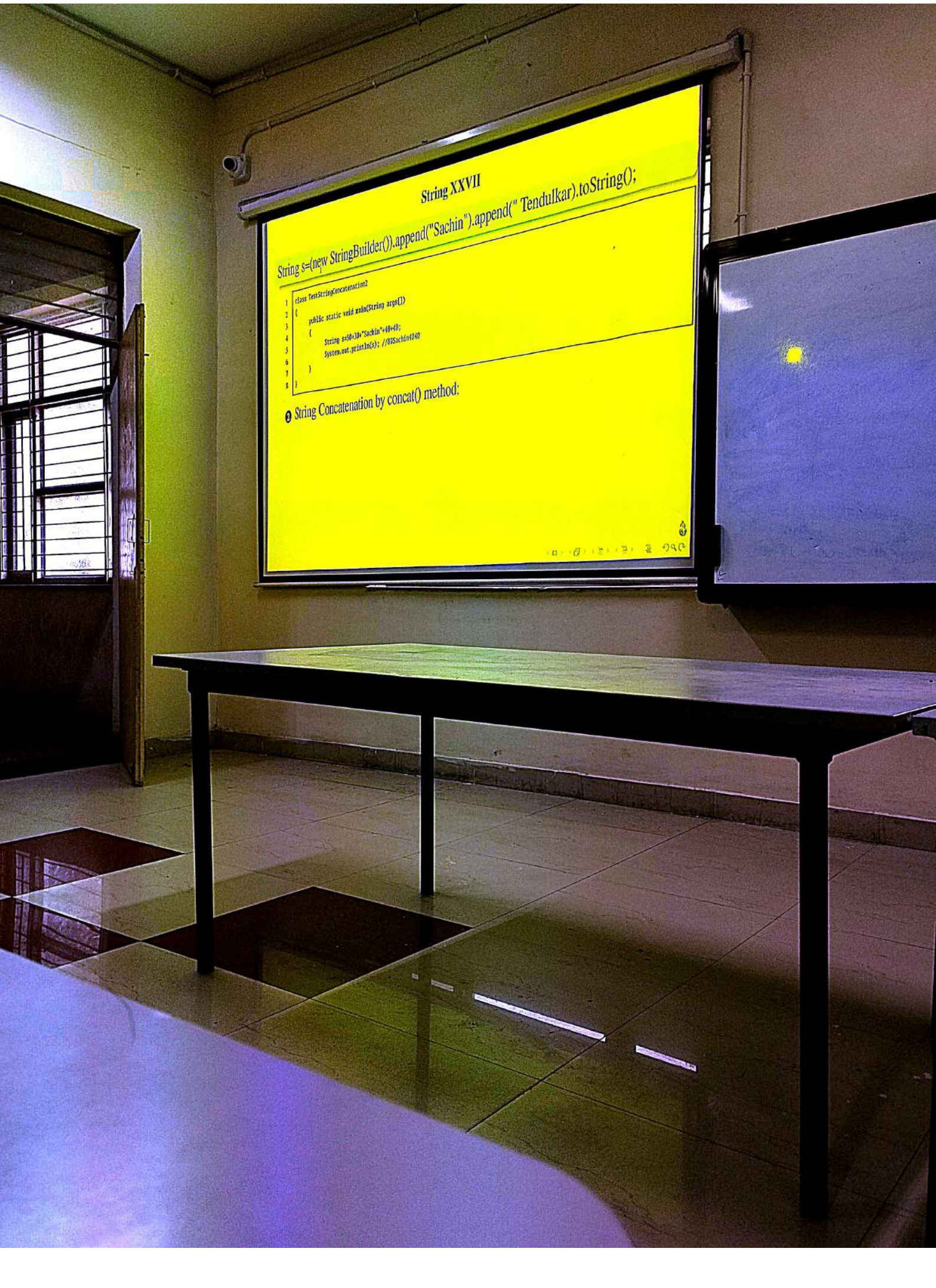
String XXV

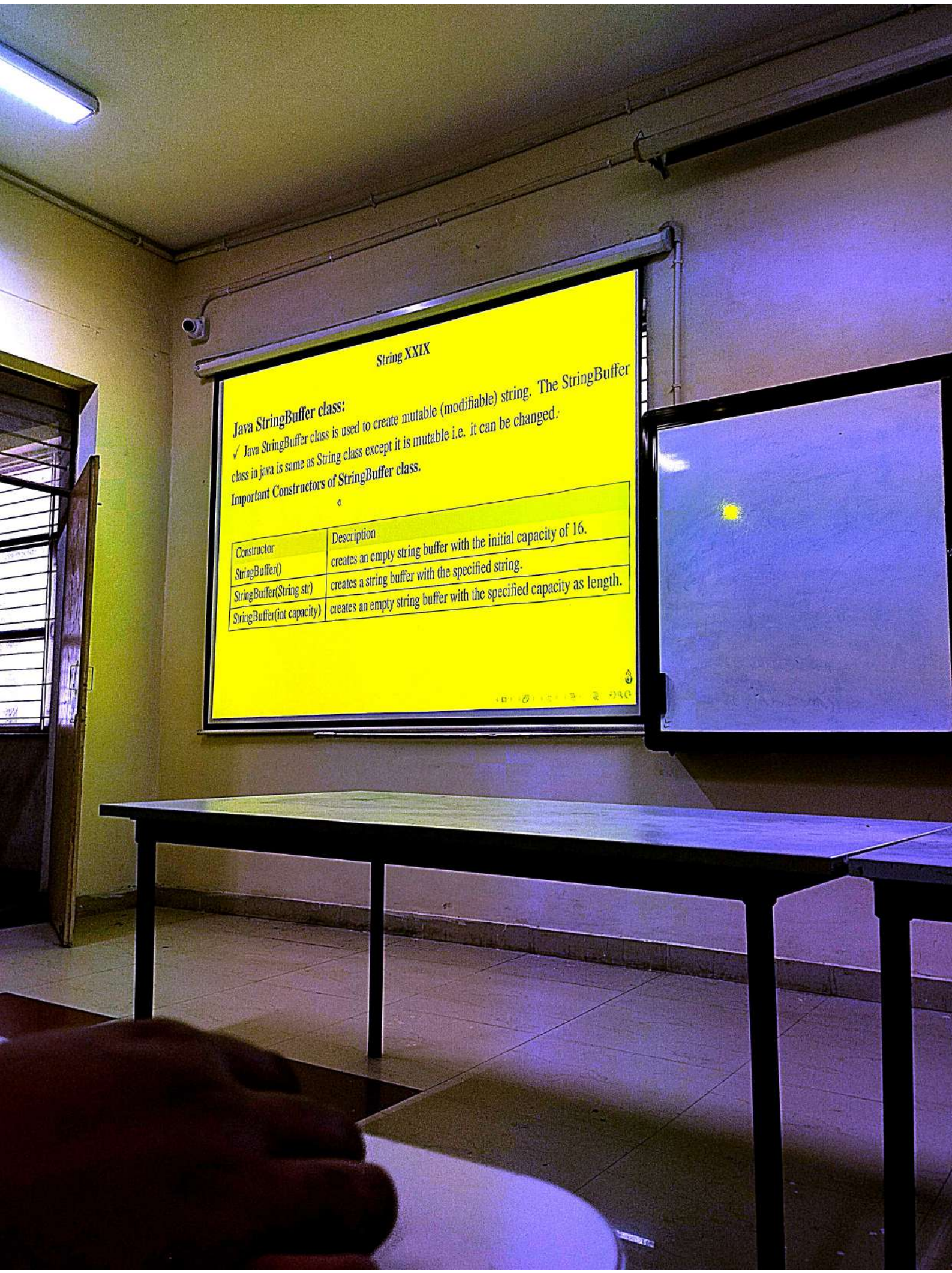
✓ But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

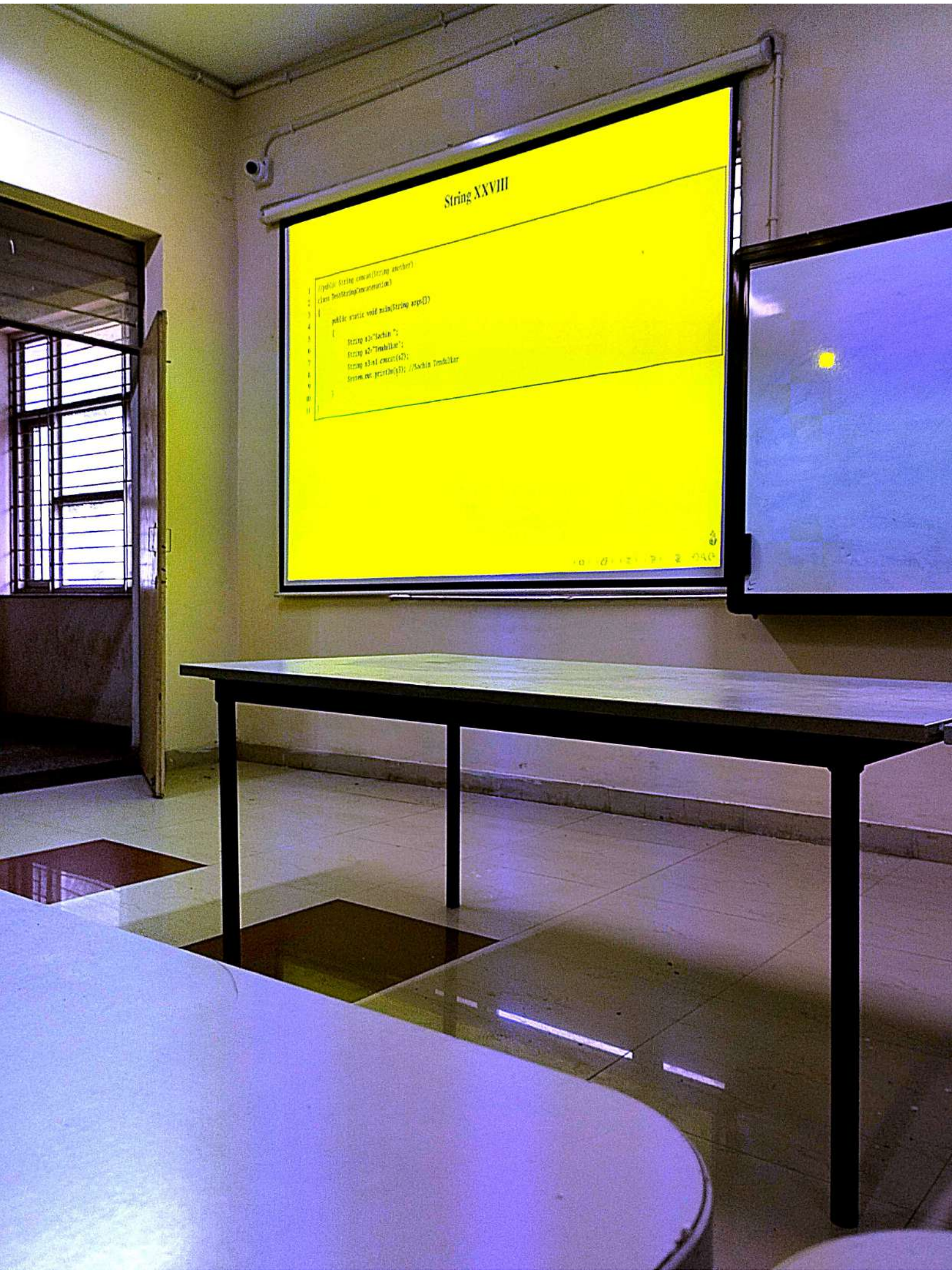
```
class Testimmutablestring1
{
    public static void main(String args[])
    {
        String s="Sachin";
        s=s.concat(" Tendulkar");
        System.out.println(s);
    }
}
```

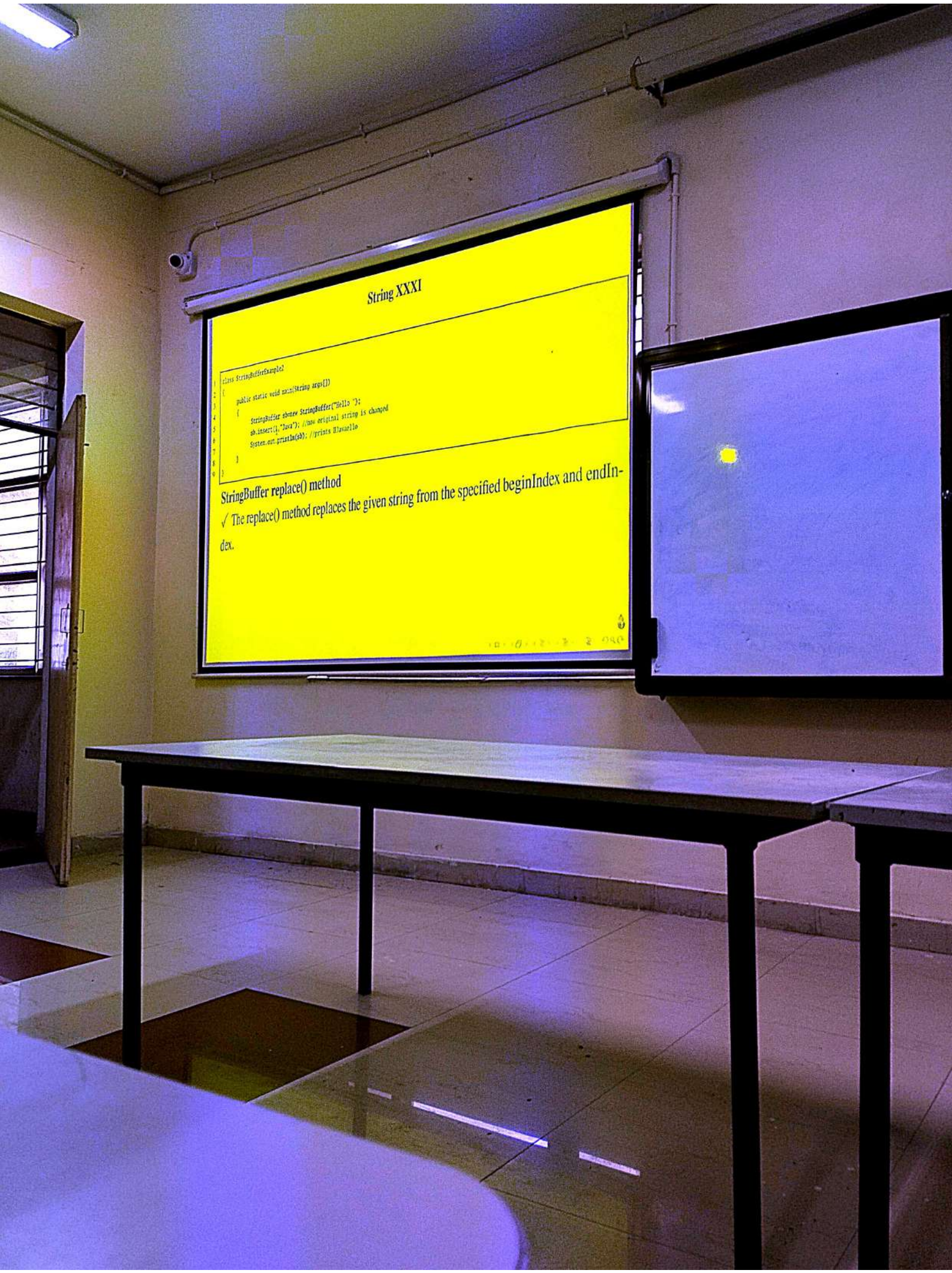
✓ Because java uses the concept of string literal.Suppose there are 5 reference variables,all refers to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

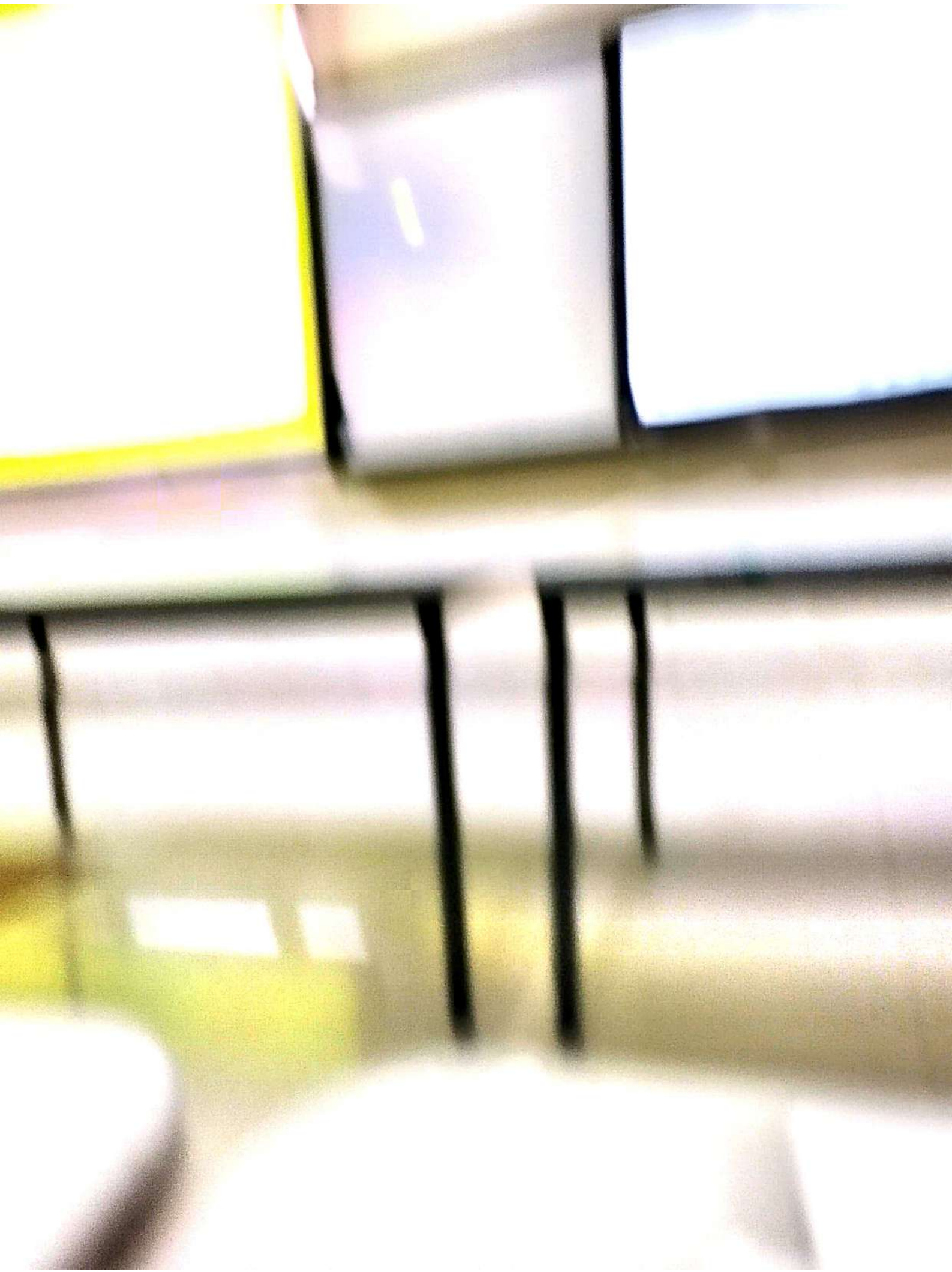

String XXX
Important methods of StringBuffer class:
StringBuffer append() method
✓ The append() method concatenates the given argument with this string.
class StringBufferExample
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java"); //now original string is changed
System.out.println(sb); //prints Hello Java
}
}
StringBuffer insert() method:
✓ The insert() method inserts the given string with this string at the given position.


String XXVII
String s=(new StringBuilder()).append("Sachin").append(" Tendulkar).toString();
class TestStringConcatenation2
{
public static void main(String args[])
{
String s=50+30+"Sachin"+40+40;
System.out.println(s); //80Sachin4040
}
}
String Concatenation by concat() method:


String XXIX
Java StringBuffer class:
✓ Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
Important Constructors of StringBuffer class.
Constructor
Description
StringBuffer()
creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)
creates a string buffer with the specified string.
StringBuffer(int capacity)
creates an empty string buffer with the specified capacity as length.


String XXVIII
//public String concat(String another)
class TestStringConcatenation
{
public static void main(String args[])
{
String s1="Sachin ";
String s2="Tendulkar";
String s3=s1.concat(s2);
System.out.println(s3); //Sachin Tendulkar
}
}


String XXXI
class StringBufferExample2
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java"); //now original string is changed
System.out.println(sb); //prints HJavaello
}
}
StringBuffer replace() method
✓ The replace() method replaces the given string from the specified beginIndex and endIndex.



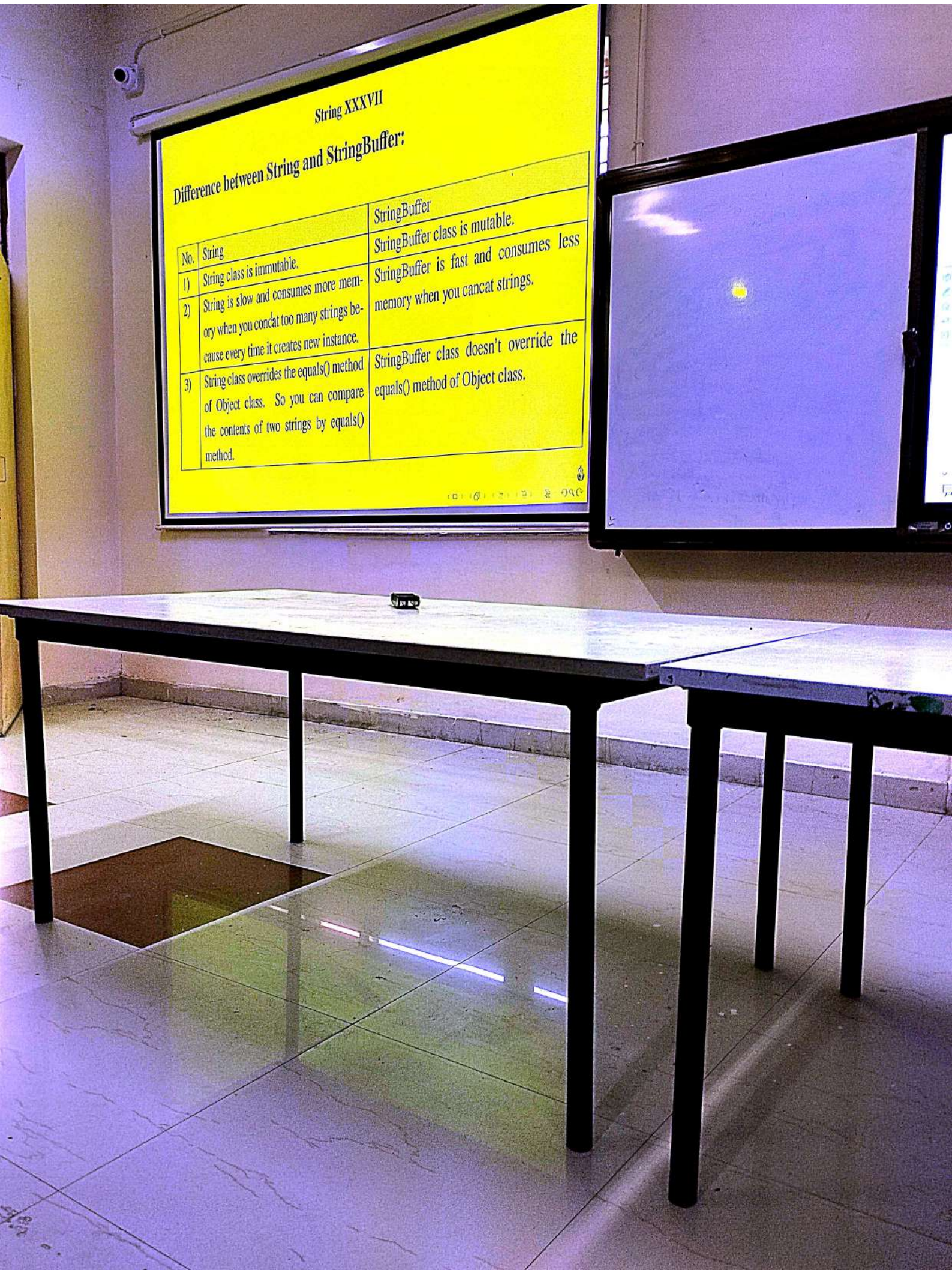

String XXXI
class StringBufferExample2
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java"); //now original string is changed
System.out.println(sb); //prints HJavaello
}
}
StringBuffer replace() method
✓ The replace() method replaces the given string from the specified beginIndex and endIn-
dex.


String XXVIII
StringBuffer reverse() method
The reverse() method of StringBuilder class reverses the current string.

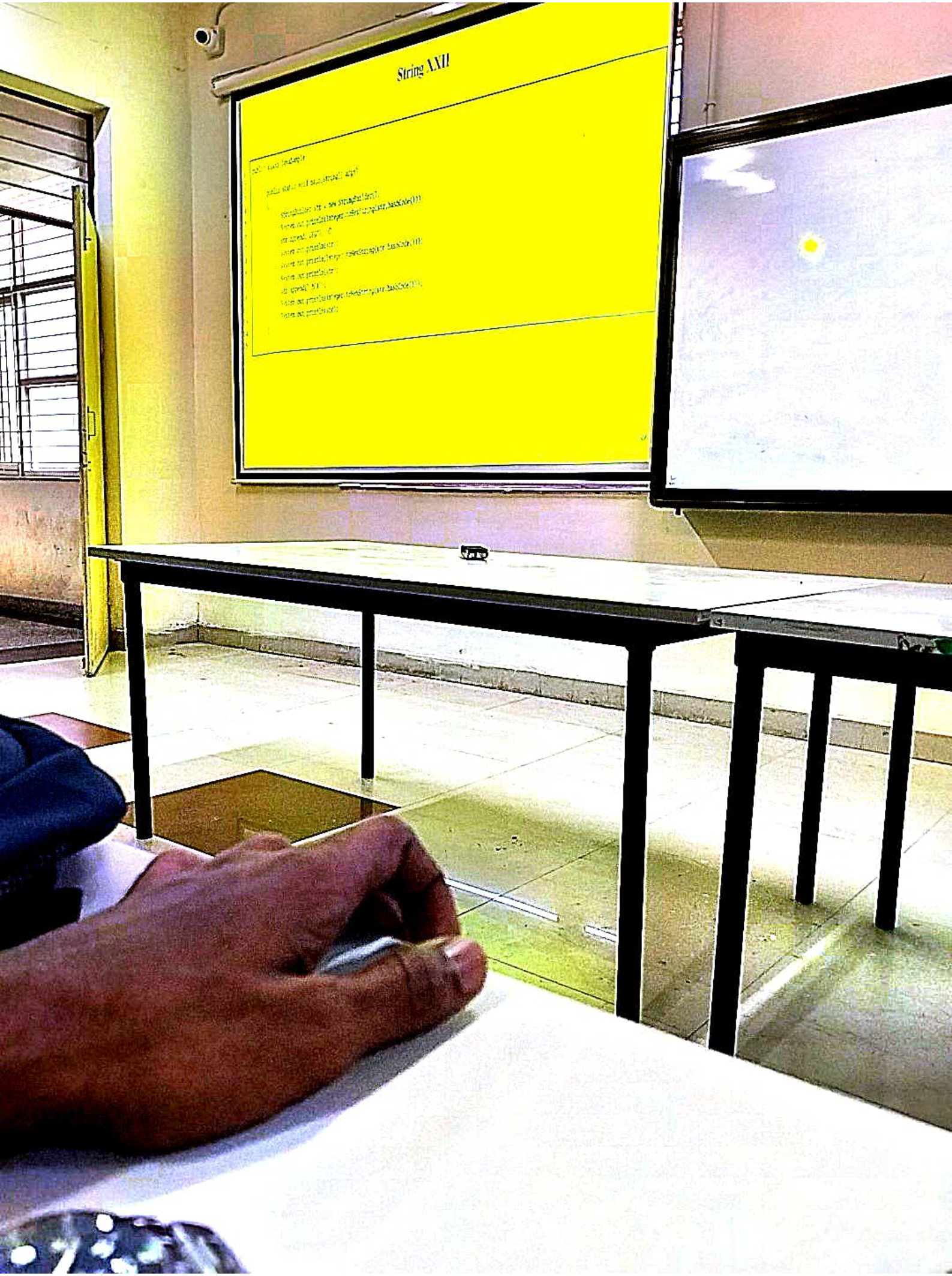

String XXXIV
StringBuffer capacity() method
The capacity() method of StringBuffer class returns the current capacity of the buffer.
The default capacity of the buffer is 16. If the number of character increases from its
current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current
capacity is 16, it will be (16*2)+2=34.


String XXXIX
long startTime = System.currentTimeMillis();
concatWithString();
System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
startTime = System.currentTimeMillis();
concatWithStringBuffer();
System.out.println("Time taken by Concating with StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms");
}


String XXXIV
StringBuffer capacity() method
√ The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.
class StringBufferExample6
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity()); //default 16
sb.append("Hello");
System.out.println(sb.capacity()); //now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity()); //now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}

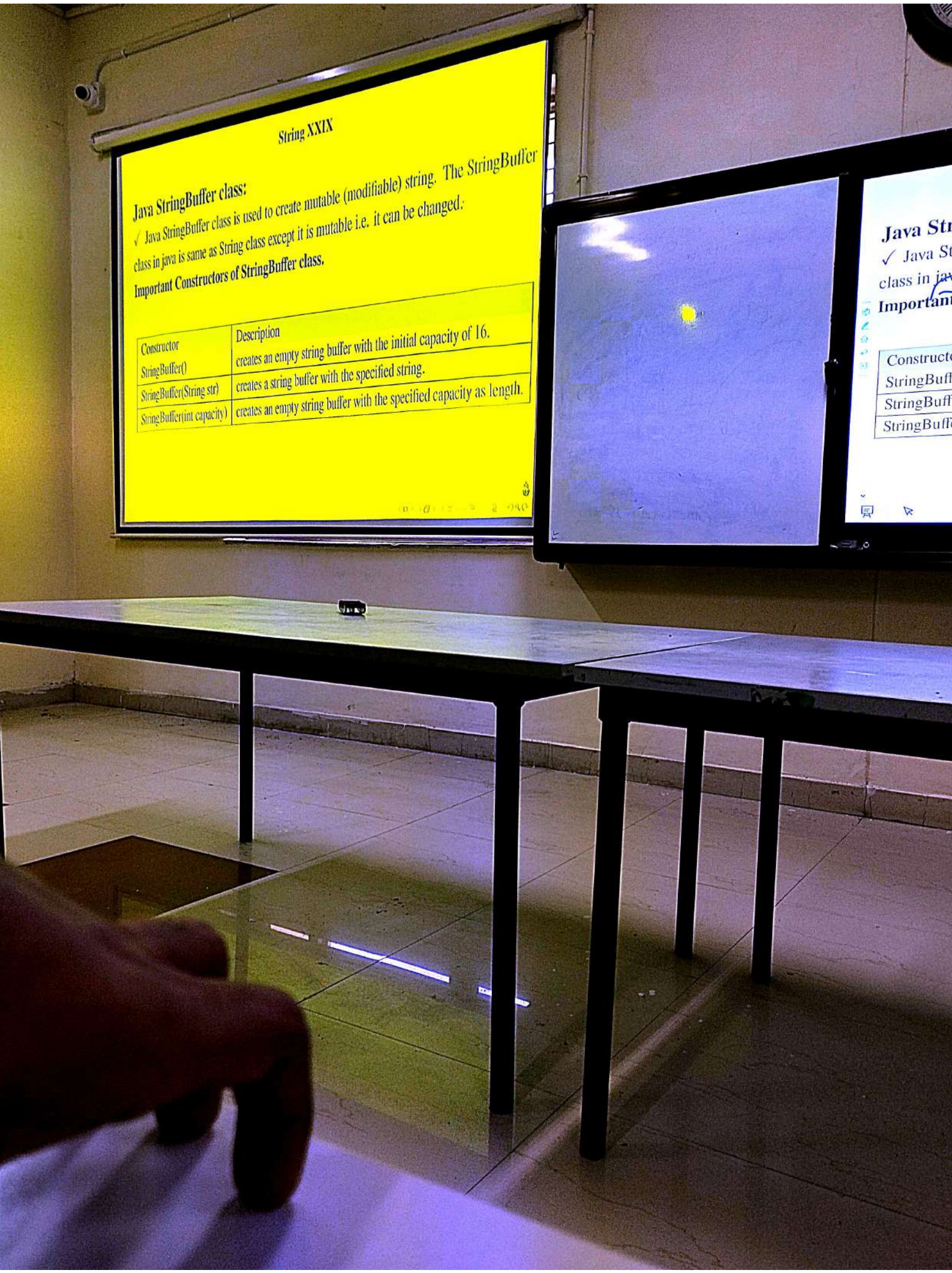

String XXXVII
Difference between String and StringBuffer:
No.
String
StringBuffer
1)
String class is immutable.
StringBuffer class is mutable.
2)
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.
StringBuffer is fast and consumes less memory when you cancat strings.
3)
String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.
StringBuffer class doesn't override the equals() method of Object class.


String XXXVIII
Performance Test of String and StringBuffer
public class ConcatTest
{
public static String concatWithString()
{
String t = "Java";
for (int i=0; i<10000; i++)
{
t = t + "Ipoint";
}
return t;
}
public static String concatWithStringBuffer()
{
StringBuffer sb = new StringBuffer("Java");
for (int i=0; i<10000; i++)
{
sb.append("Ipoint");
}
return sb.toString();
}
public static void main(String[] args)
{


String AXII


String XLI
Performance Test of StringBuffer and StringBuilder:
//Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.
public class ConcatTest
{
public static void main(String[] args)
{
long startTime = System.currentTimeMillis();
StringBuffer sb = new StringBuffer("Java");
for (int i=0; i<10000; i++)
{
sb.append("Tpoint");
}
System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
startTime = System.currentTimeMillis();
StringBuilder sb2 = new StringBuilder("Java");
for (int i=0; i<10000; i++)
{
sb2.append("Tpoint");
}
System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
}
}


String XL
Difference between StringBuffer and StringBuilder:
StringBuffer
vs
StringBuilder
StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously
StringBuffer is less efficient than StringBuilder
StringBuilder is more efficient than StringBuffer.


String XXIX
Java StringBuffer class:
✓ Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
Important Constructors of StringBuffer class.
Constructor
Description
StringBuffer()
creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)
creates a string buffer with the specified string.
StringBuffer(int capacity)
creates an empty string buffer with the specified capacity as length.


Solution (cont..)