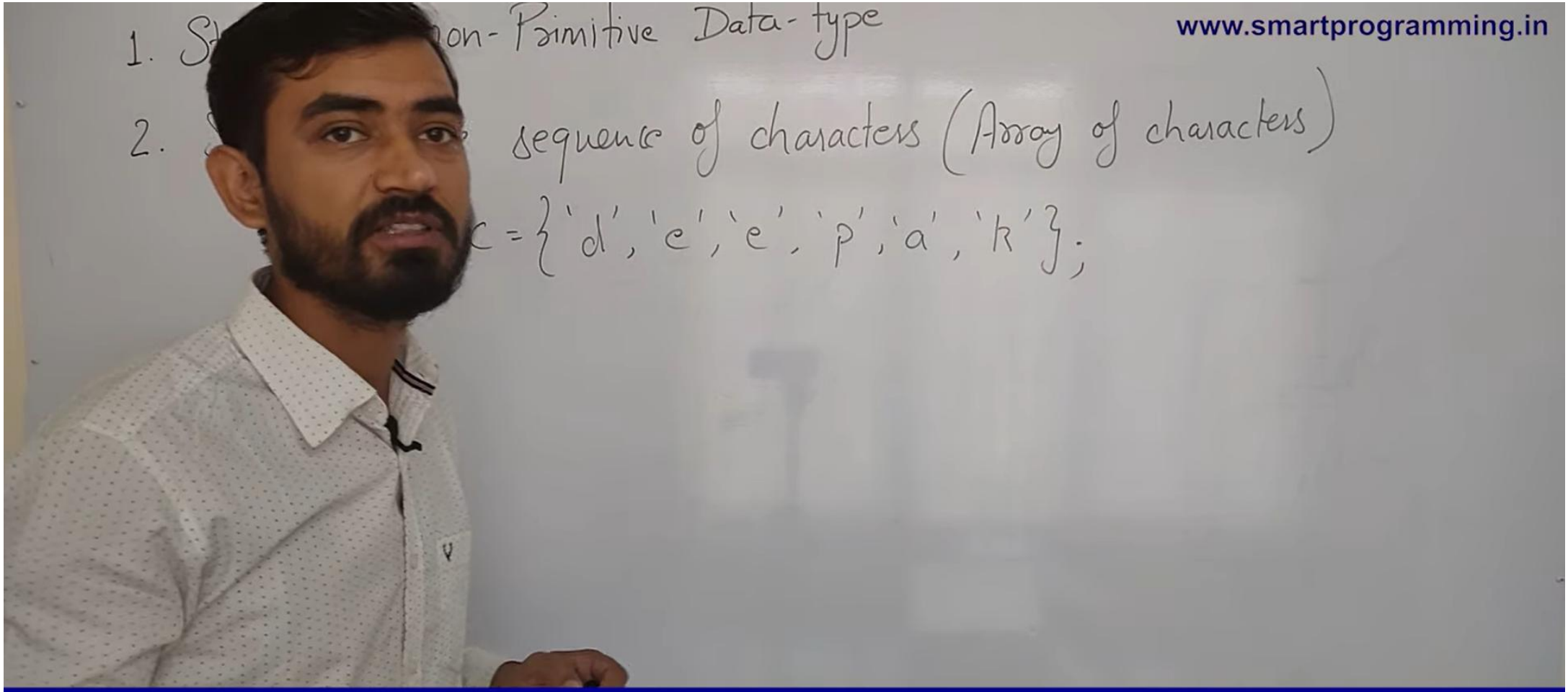
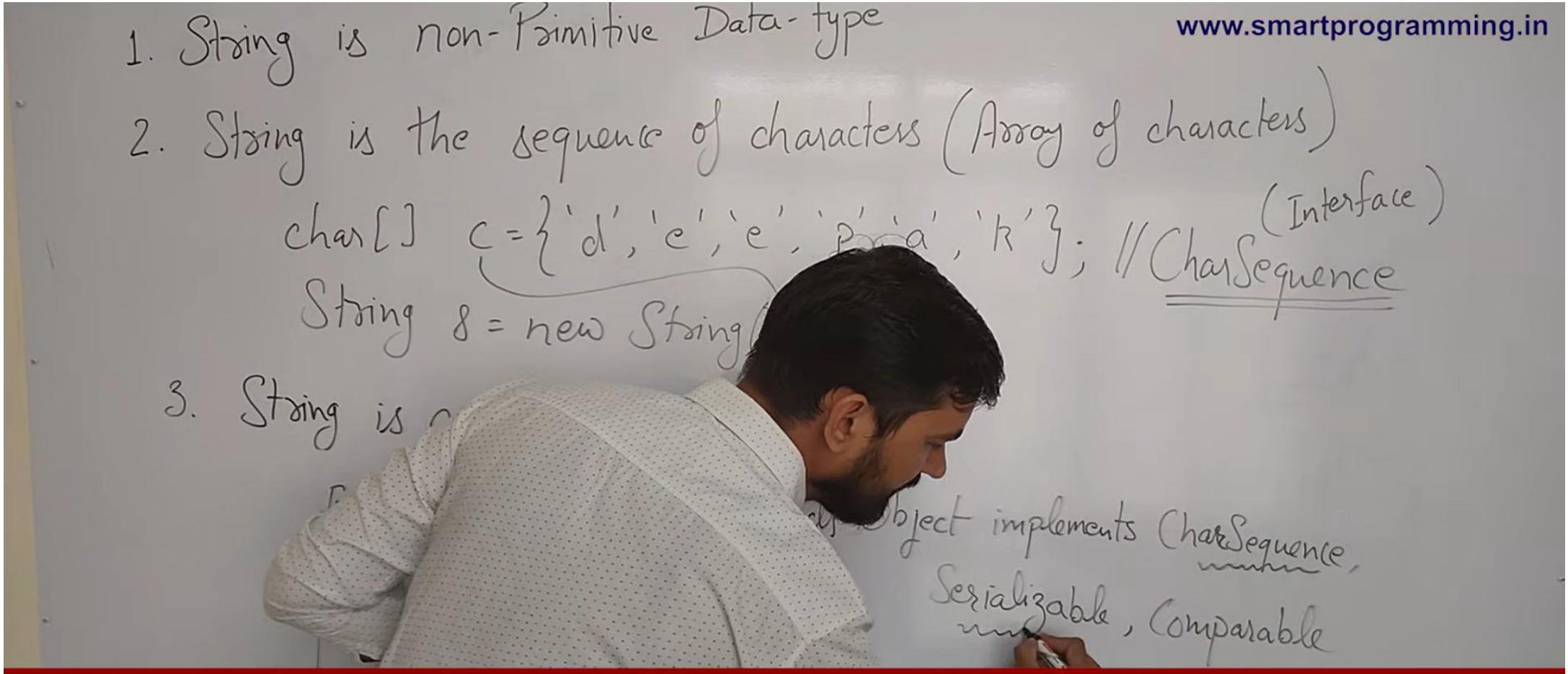
String is a non-primitive data types because it references a memory location where data is stored in the heap memory (or String Constant Pool) i.e, it references to a memory where an object is actually placed. And thus the variable of a non-primitive data type is also called reference data types or object reference variable. This object reference variable lives on the stack memory and the object to which it points always lives on the heap memory. The stack holds a pointer to the object on the heap.
Thus all non-primitive data types are simply called objects which are created by instantiating a class.

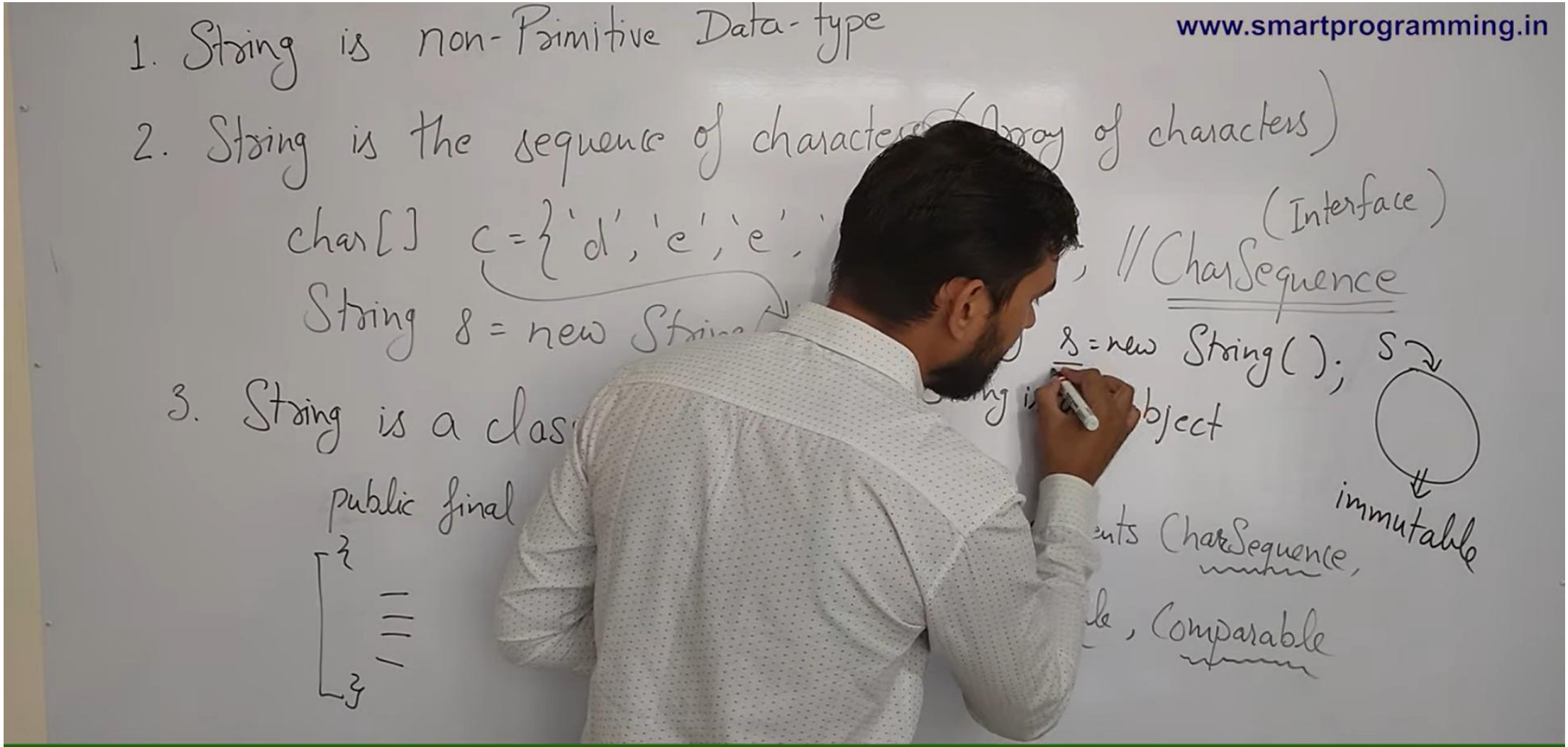


String is the sequence of characters or say, String is an array of characters. For eg.
-> char[] c={'d','e','e','p','a','k'};
-> String s=new String(c);
is same as.... String s="deepak";

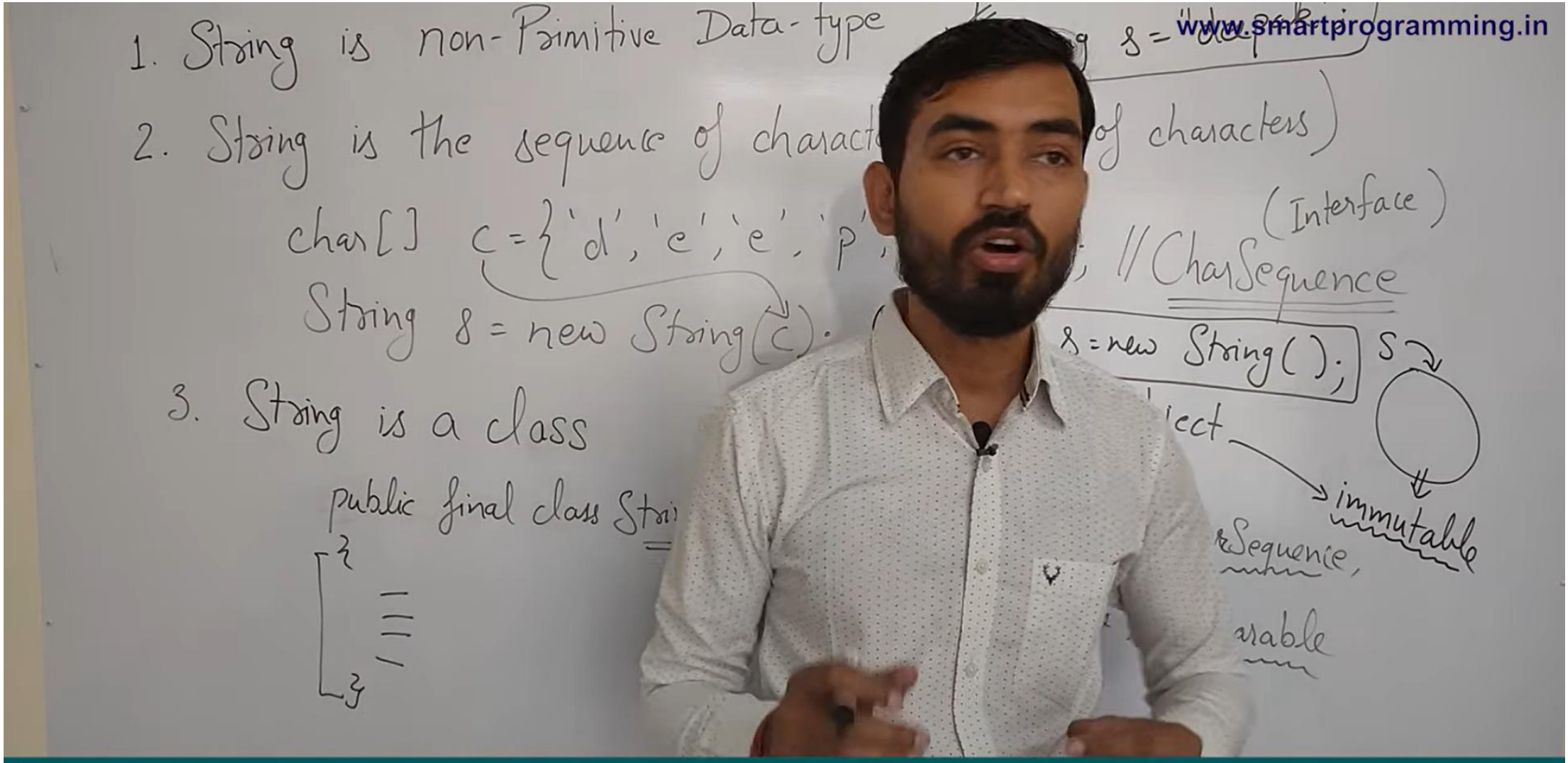


Serializable interface is used when we need to store a copy of the object and send them to another process which runs on the same system or over the network.

Comparable interface is used to order the objects of the class. It has only one method i.e. compareTo()



We can create String class object by :- String s=new String();
It will create an IMMUTABLE object.



To create String, there are three classes:-

1. String 2. StringBuffer 3. StringBuilder

String is non-Primitive Data-type ✓ String s = "deepak";

String is the sequence of characters (Array of characters)

char[] c = {'d','e','e','p','a','k'}; // CharSequence (Interface)

String s = new String(c);

① String s = new String(); → s → immutable

String is an object

String is a class

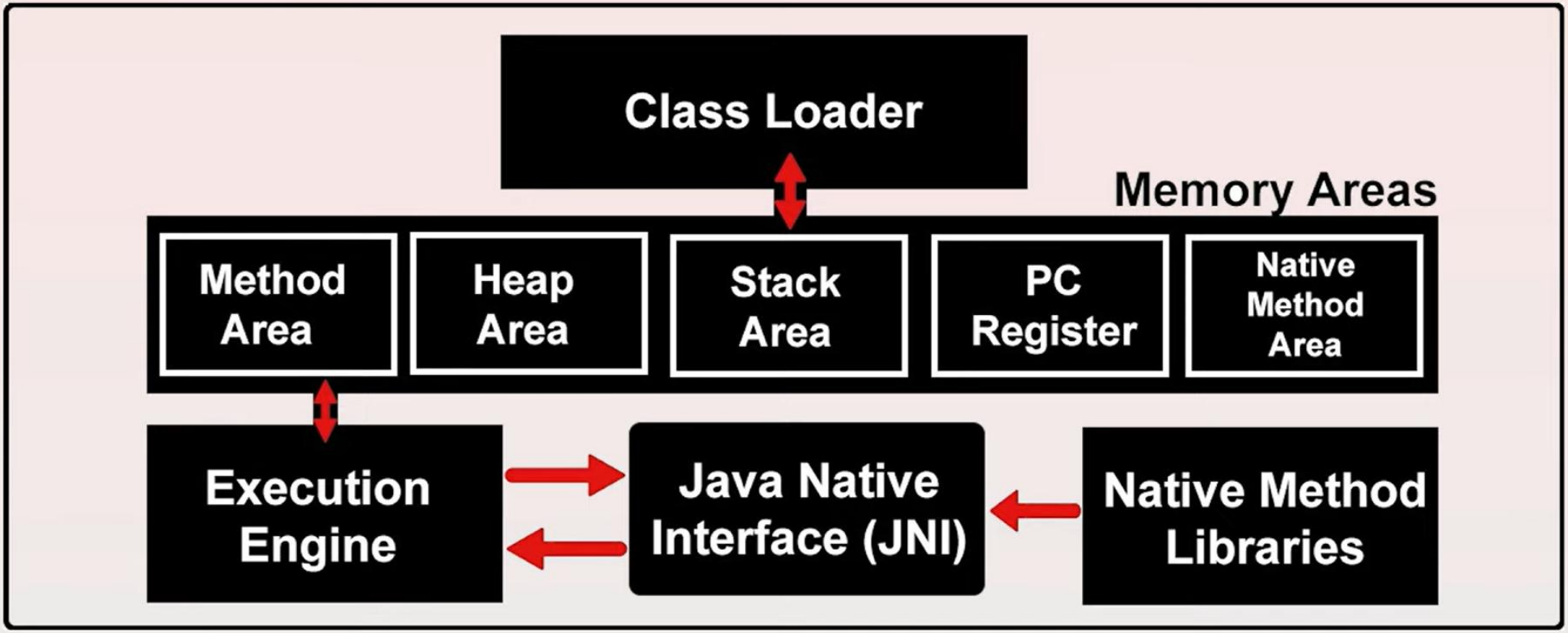
public final class String extends Object implements CharSequence, Serializable, Comparable

{ }

⑤ To create Strings, there are 3 classes
→ String → StringBuffer → StringBuilder

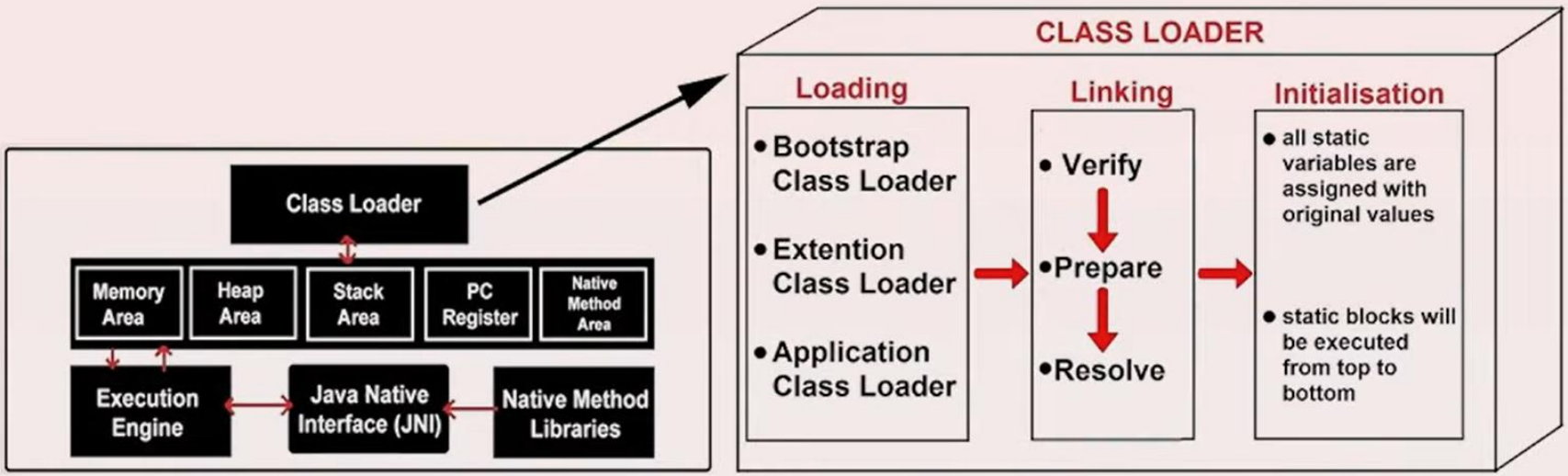

Strings Introduction in Java by Deepak (Hindi)
String Constant Pool (or String Literal Pool) is an area in heap memory where java stores String literal values.

JVM Architecture



JVM Architecture (Class Loader Diagram)

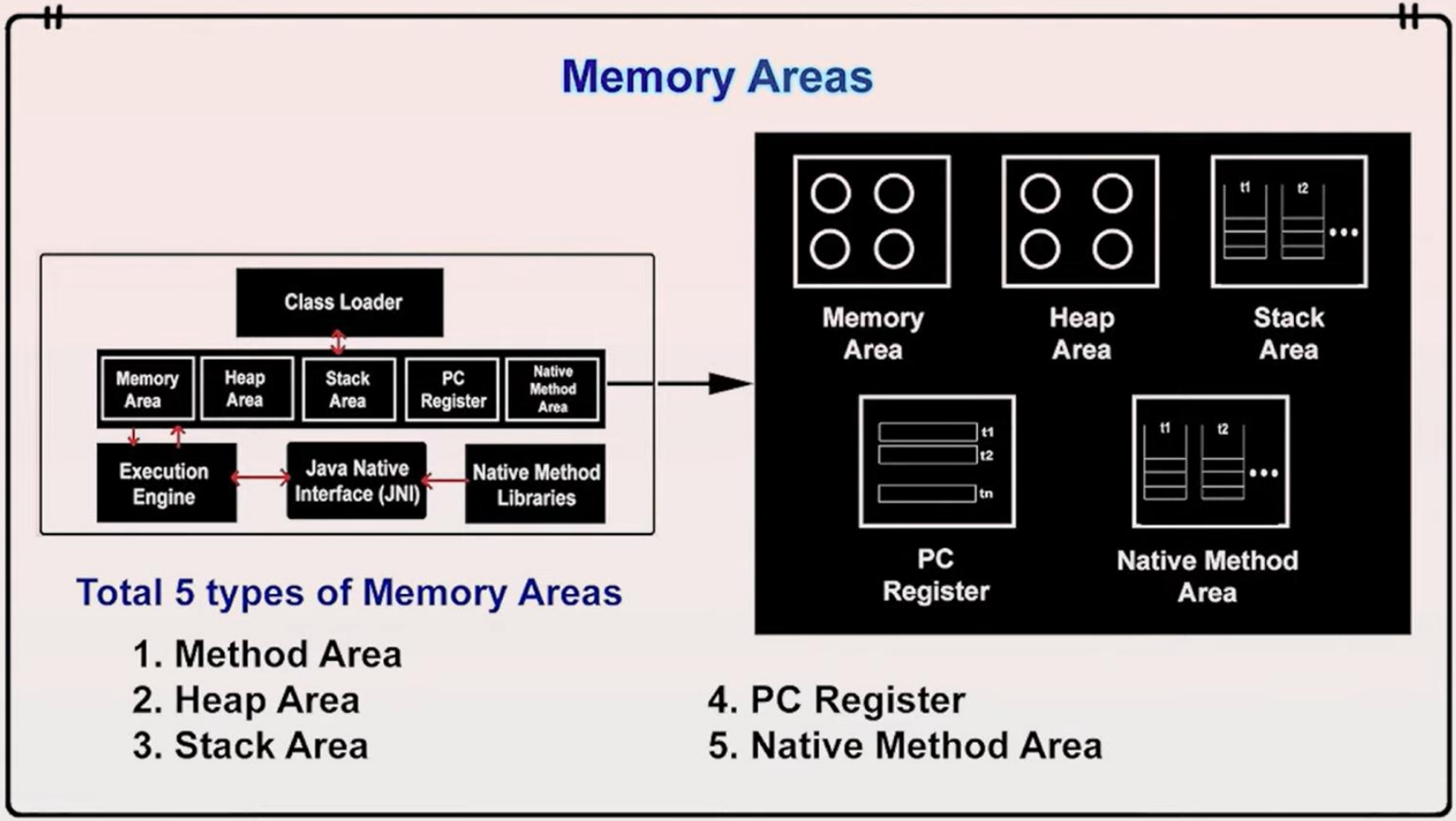
Class Loader SubSystem



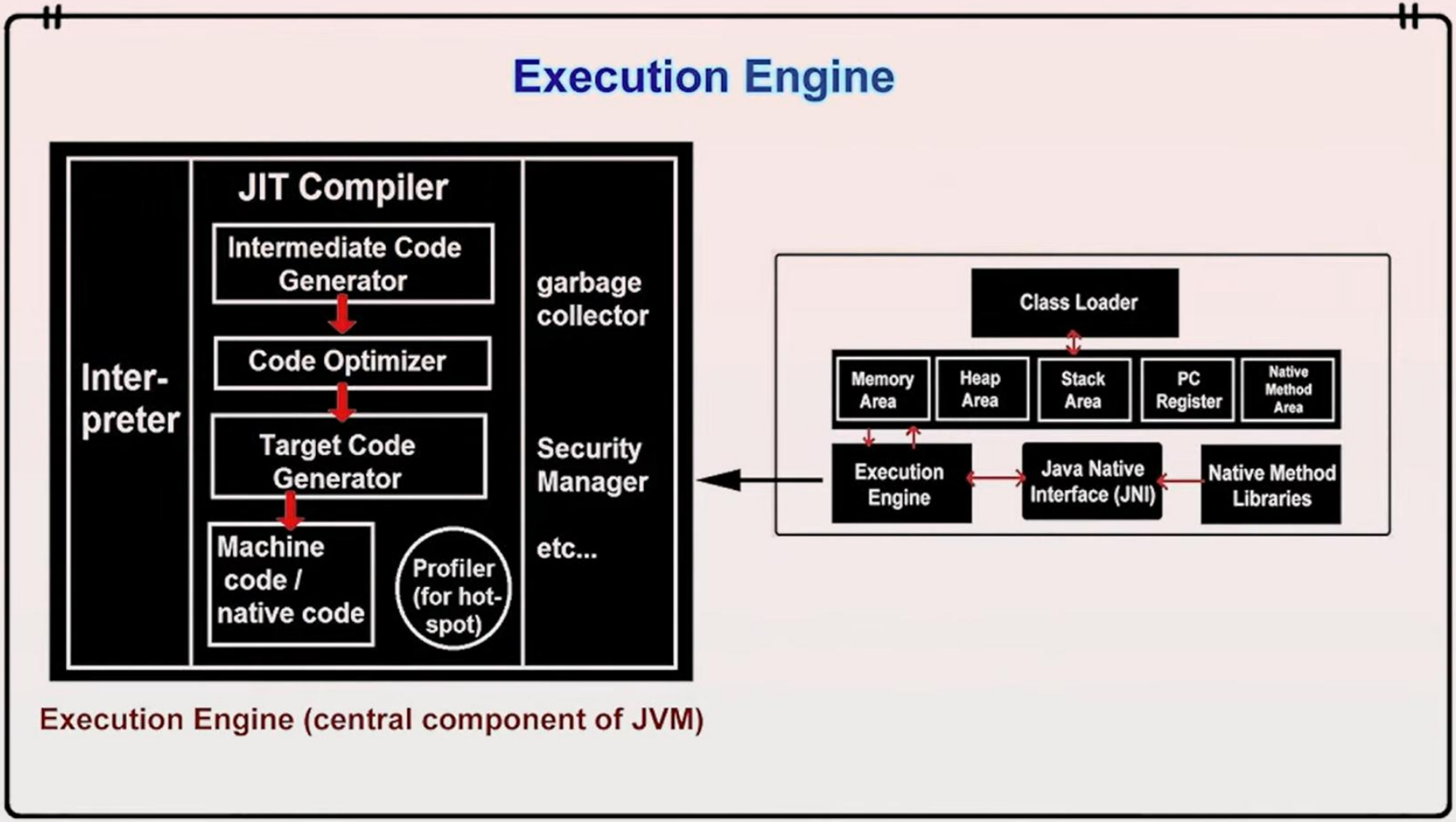
Class Loader : It is responsible for the following three tasks :

1. Loading
2. Linking
3. Initializing

JVM Architecture (Memory Areas Diagram)



JVM Architecture (Execution Engine Diagram)



Execution Engine (central component of JVM)

C:\Users\Deepak\Desktop\Test.java - Notepad++

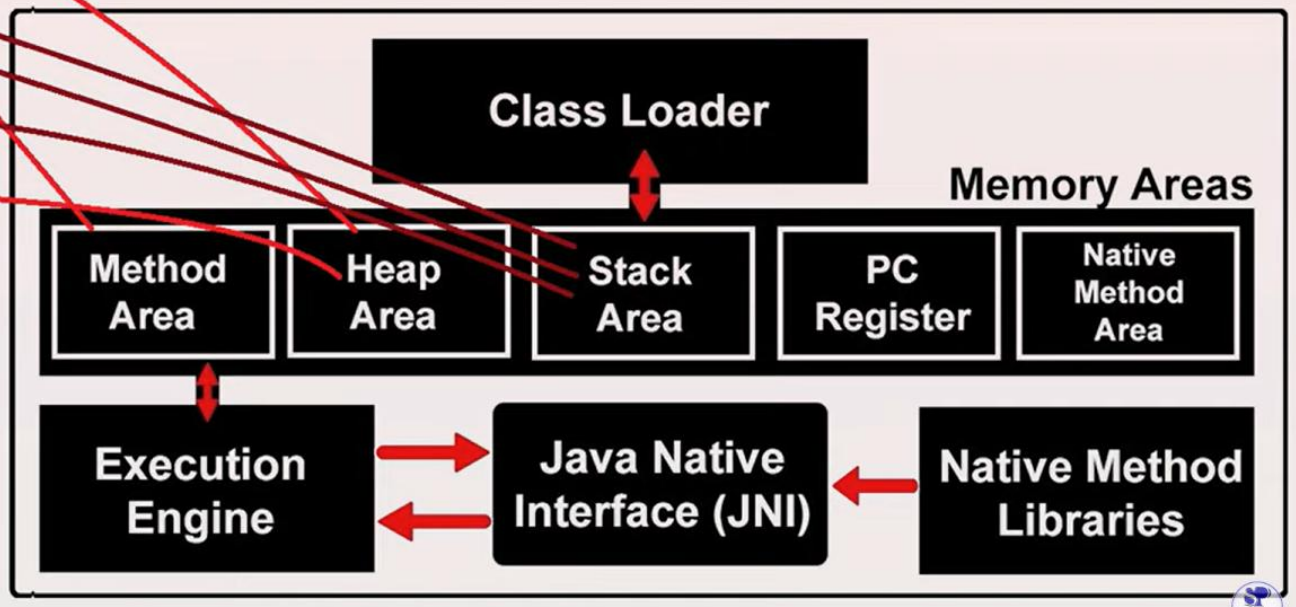
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Test.java

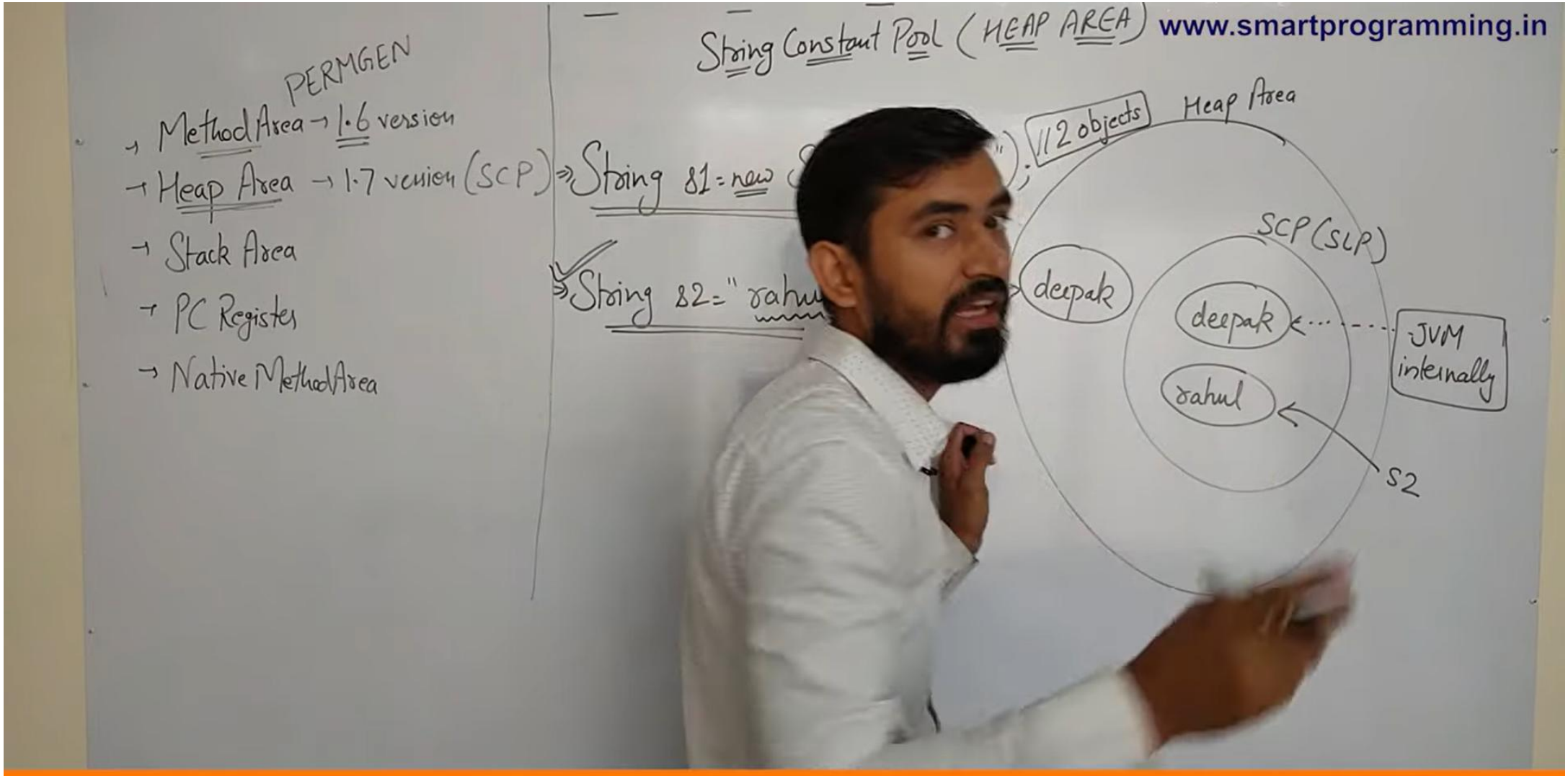
```
1  class Test
2  {
3      int a=10;
4      static int b=20;
5
6      void show()
7      {
8          int c=30;
9          //other statements
10     }
11
12     public static void main(String[] args)
13     {
14         Test t=new Test();
15         t.show();
16     }
   }
```

Java source file length : 191 lines : 16 Ln : 10 Col : 6 Sel : 0 | 0

- int a=10 (instance variable) :- It will store in Heap Area
- static int b=20 (static variable) :- It will store in Method Area
- int c=30 (local variable) :- It will store in Stack Area
- main() :- It will store in Stack Area
- Test t=new Test() (object) :- It will store in Heap Area
- t.show() (current running method) :- It will store in Stack Area



Memory Storage In Java



The String objects present in SCP are not applicable for Garbage Collection because a reference variable internally is maintained by JVM.

PERMGEN

- Method Area → 1.6 version
- Heap Area → 1.7 ver
- Stack Area
- PC Register
- Native Meth

String Constant Pool (HEAP AREA)

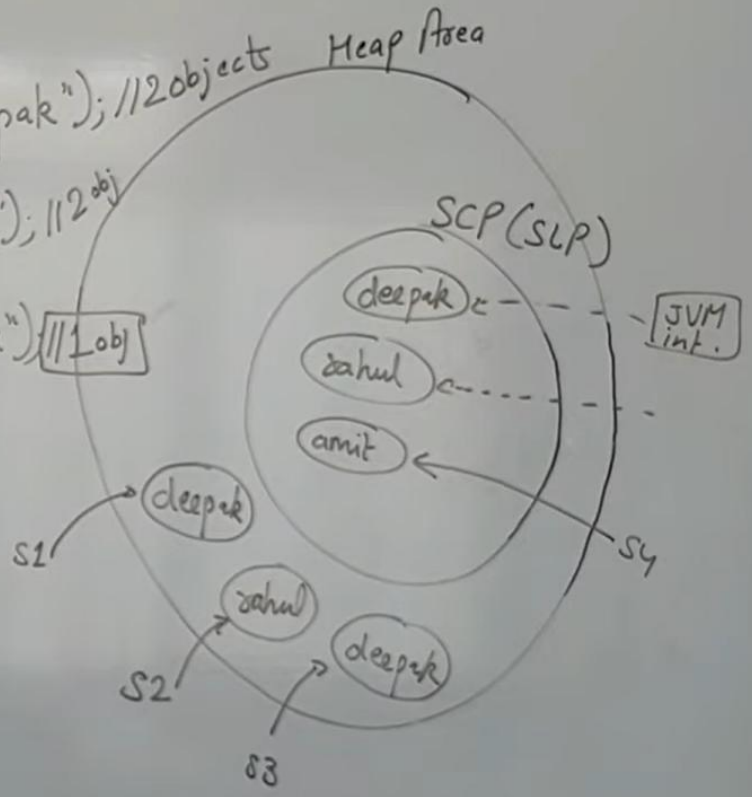
String s1 = new String("deepak"); // 2 objects

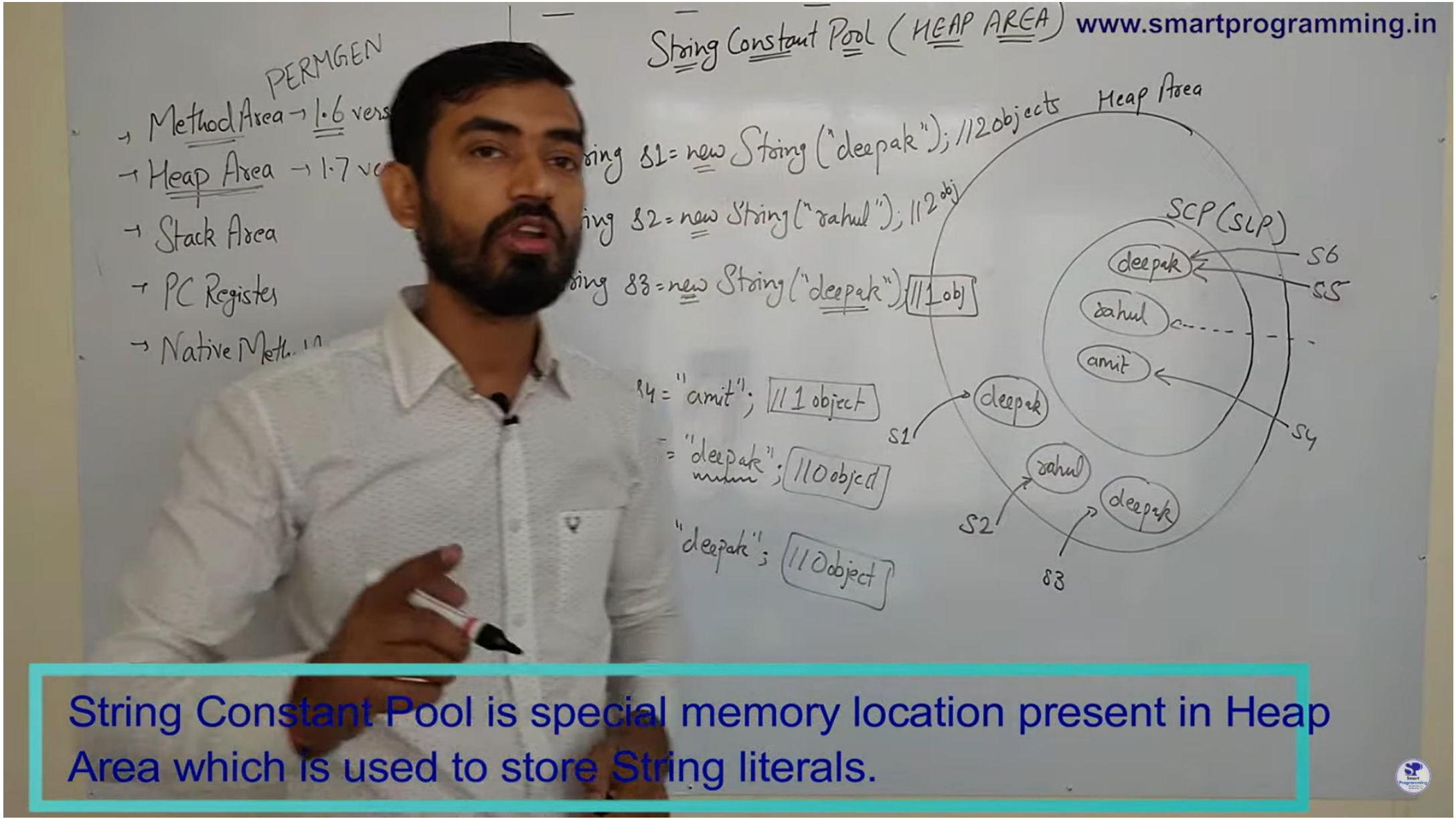
String s2 = new String("rahul"); // 2 obj

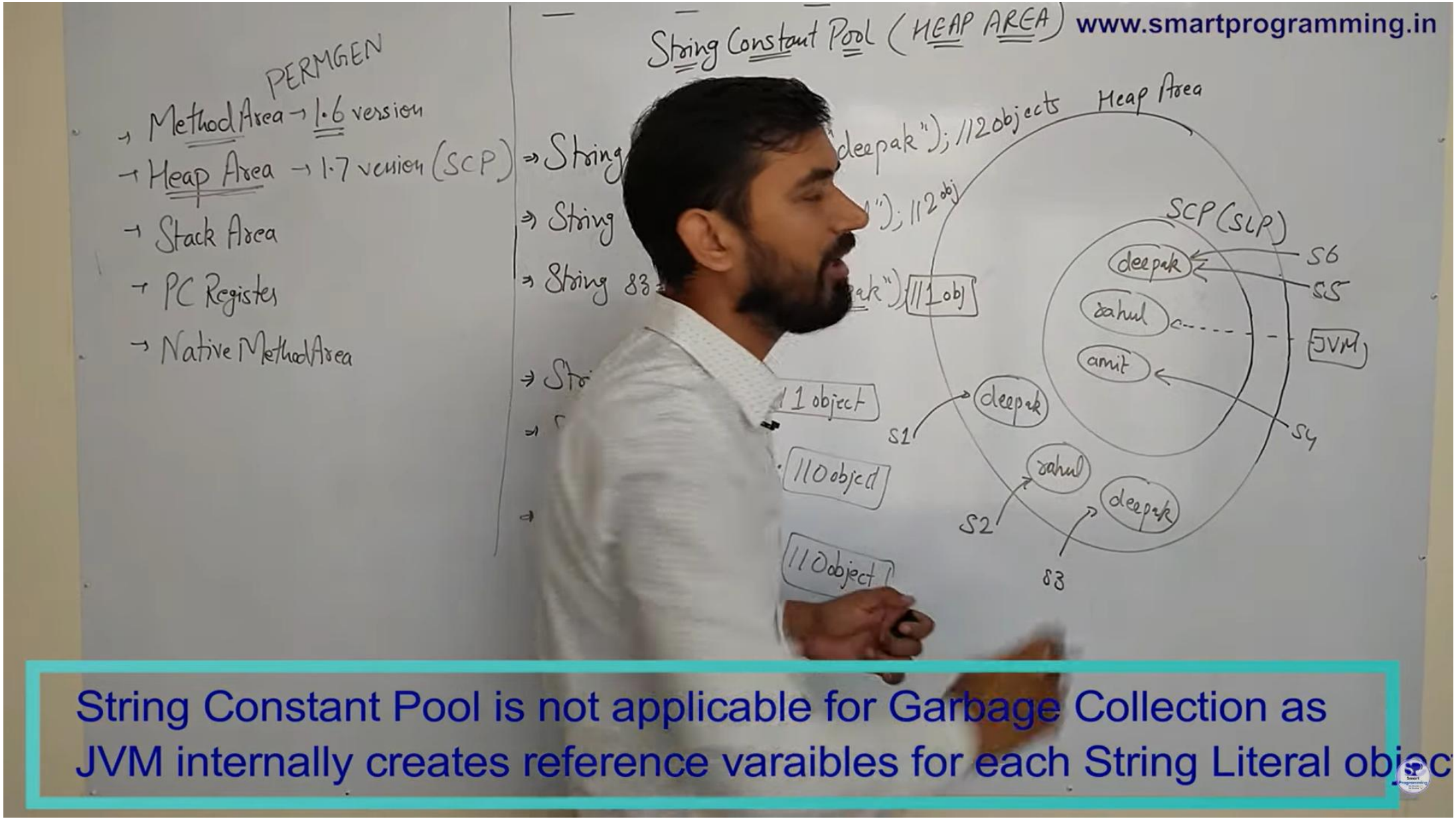
String s3 = new String("deepak"); // 1 obj

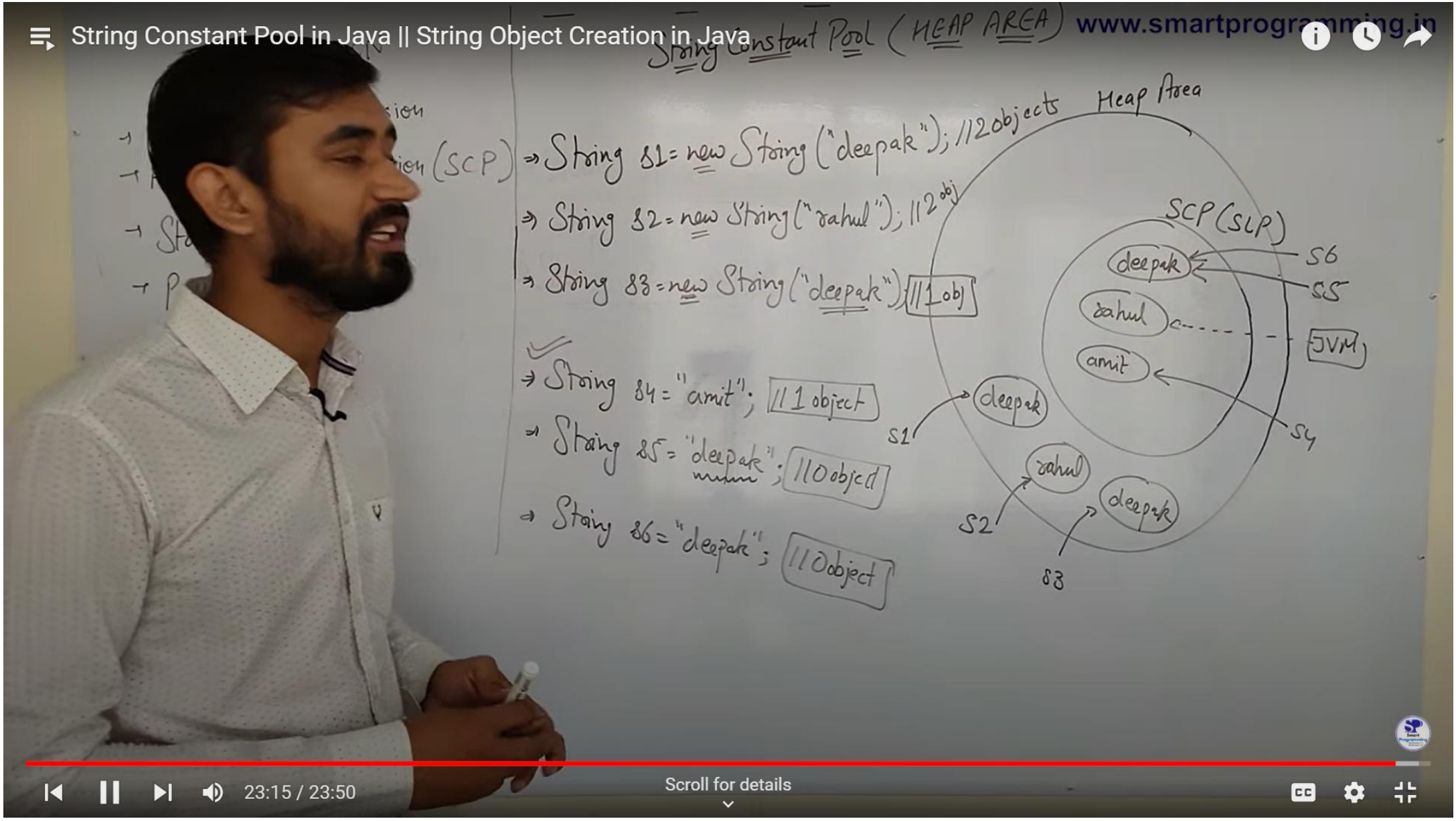
String s4 = "amit"; // 1 object

String s5 = "deepak";




String Constant Pool (HEAP AREA)
www.smartprogramming.in
PERMGEN
→ Method Area → 1.6 vers
→ Heap Area → 1.7 v
→ Stack Area
→ PC Register
→ Native Meth
ring s1 = new String ("deepak"); //2objects
ring s2 = new String ("rahul"); //2 obj
ring s3 = new String ("deepak"); //1 obj
s4 = "amit"; //1 object
= "deepak"; //0 object
"deepak"; //0 object
Heap Area
SCP (SLP)
deepak
rahul
amit
deepak
rahul
deepak
s1
s2
s3
s4
s5
s6
String Constant Pool is special memory location present in Heap Area which is used to store String literals.


PERMGEN
→ Method Area → 1.6 version
→ Heap Area → 1.7 version (SCP)
→ Stack Area
→ PC Register
→ Native Method Area
String Constant Pool (HEAP AREA)
www.smartprogramming.in
Heap Area
SCP (SLP)
deepak
rahul
amit
JVM
S1
S2
S3
S4
S5
S6
1 object
2 objects
String Constant Pool is not applicable for Garbage Collection as
JVM internally creates reference varaibles for each String Literal objec

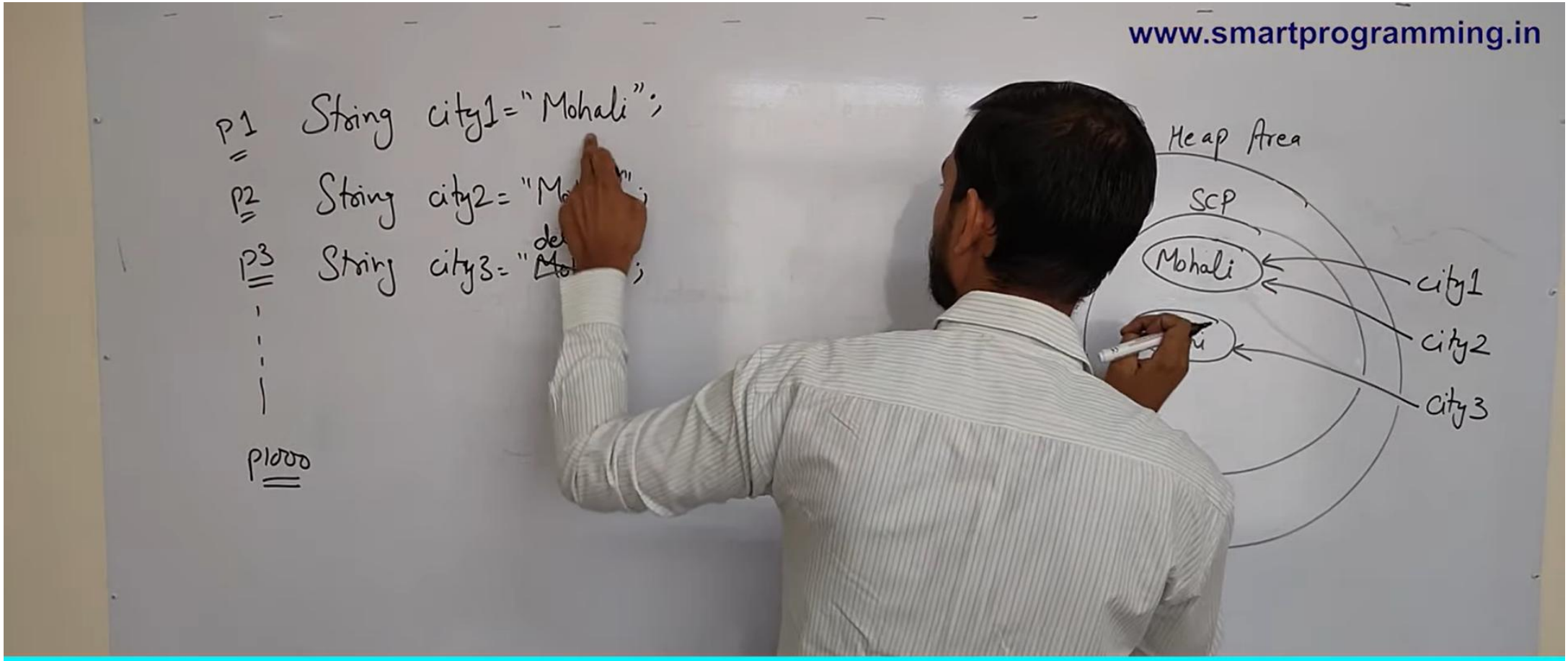


String Constant Pool in Java || String Object Creation in Java

23:15 / 23:50

Scroll for details

Immutability concept is used for "String Objects" i.e. String objects are immutable. It means once String Object is created; its data or state can't be changed but a new string object is created.



Strings are Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple persons there is always a risk, where one persons's action would affect all another persons For example, if one person changes its city from "Mohali" to "Delhi", all other persons will also get affected.

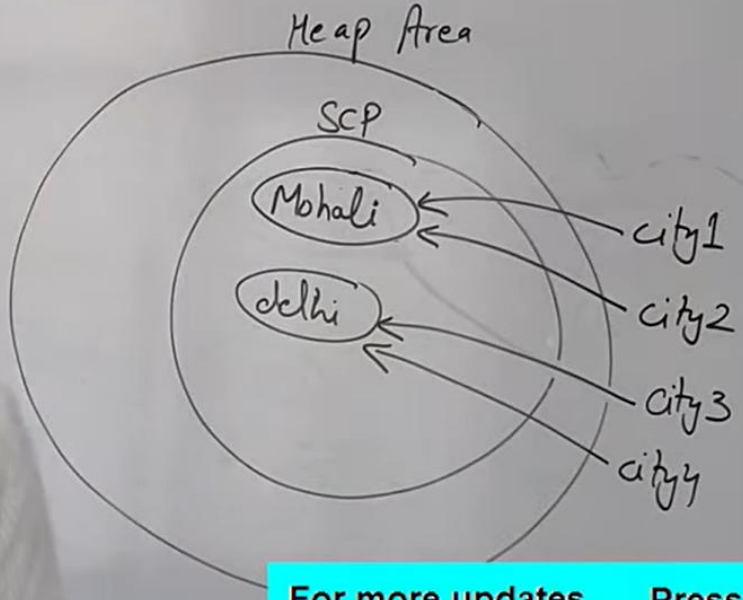
P1 String city1="Mohali";

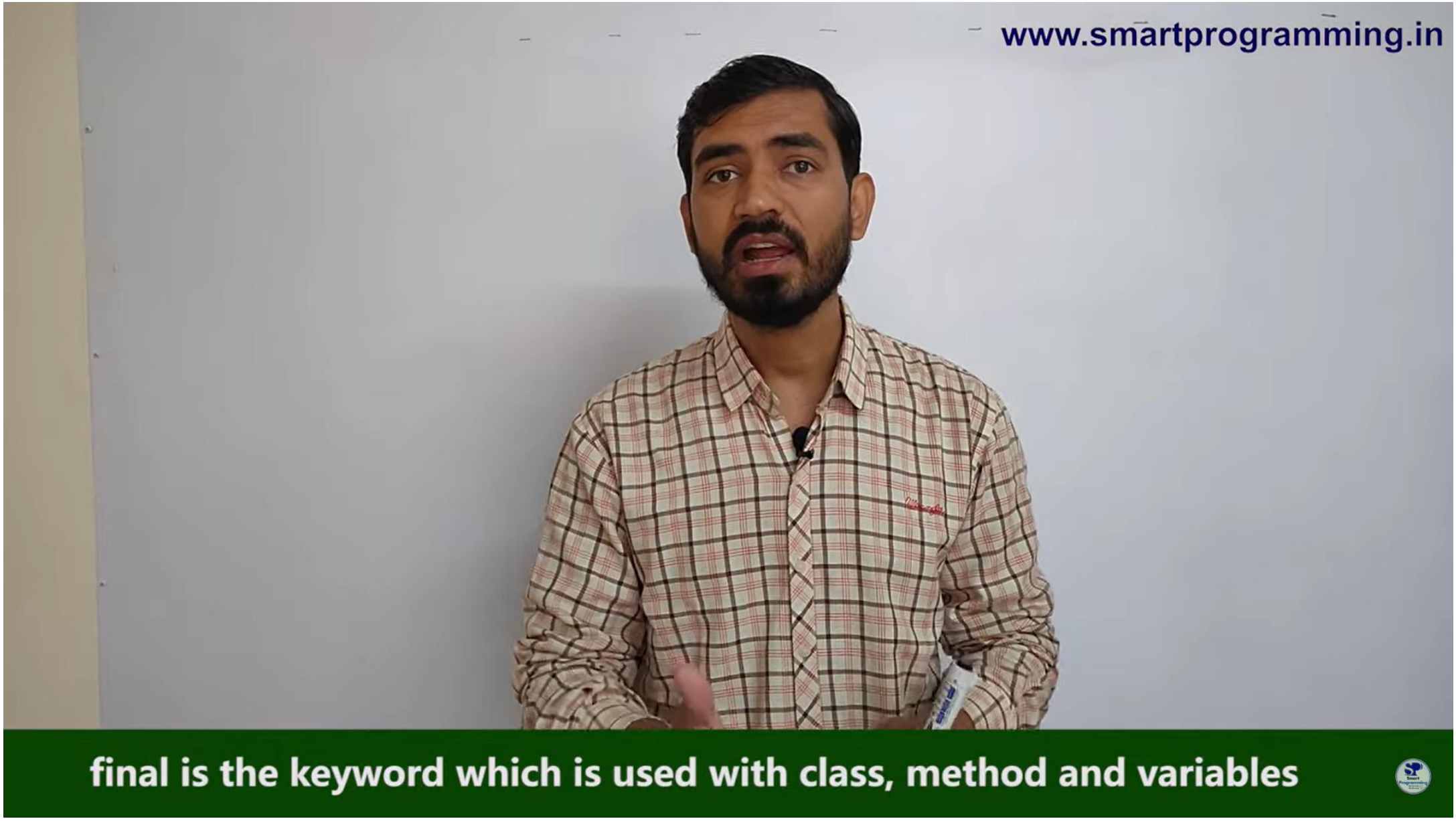
P2 String city2="Mohali";

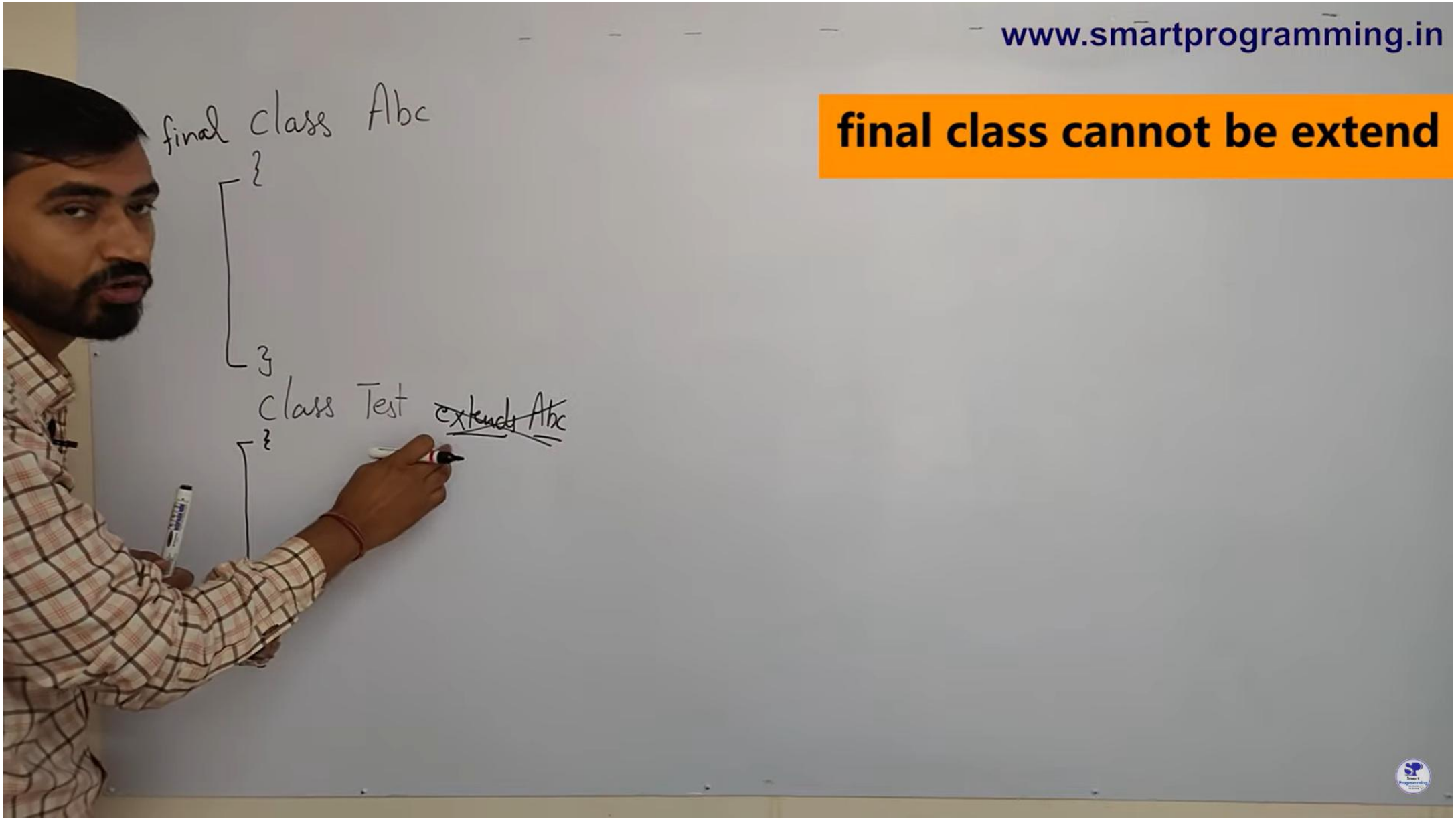
P3 String city3="~~Mohali~~ delhi";

String city4="delhi";

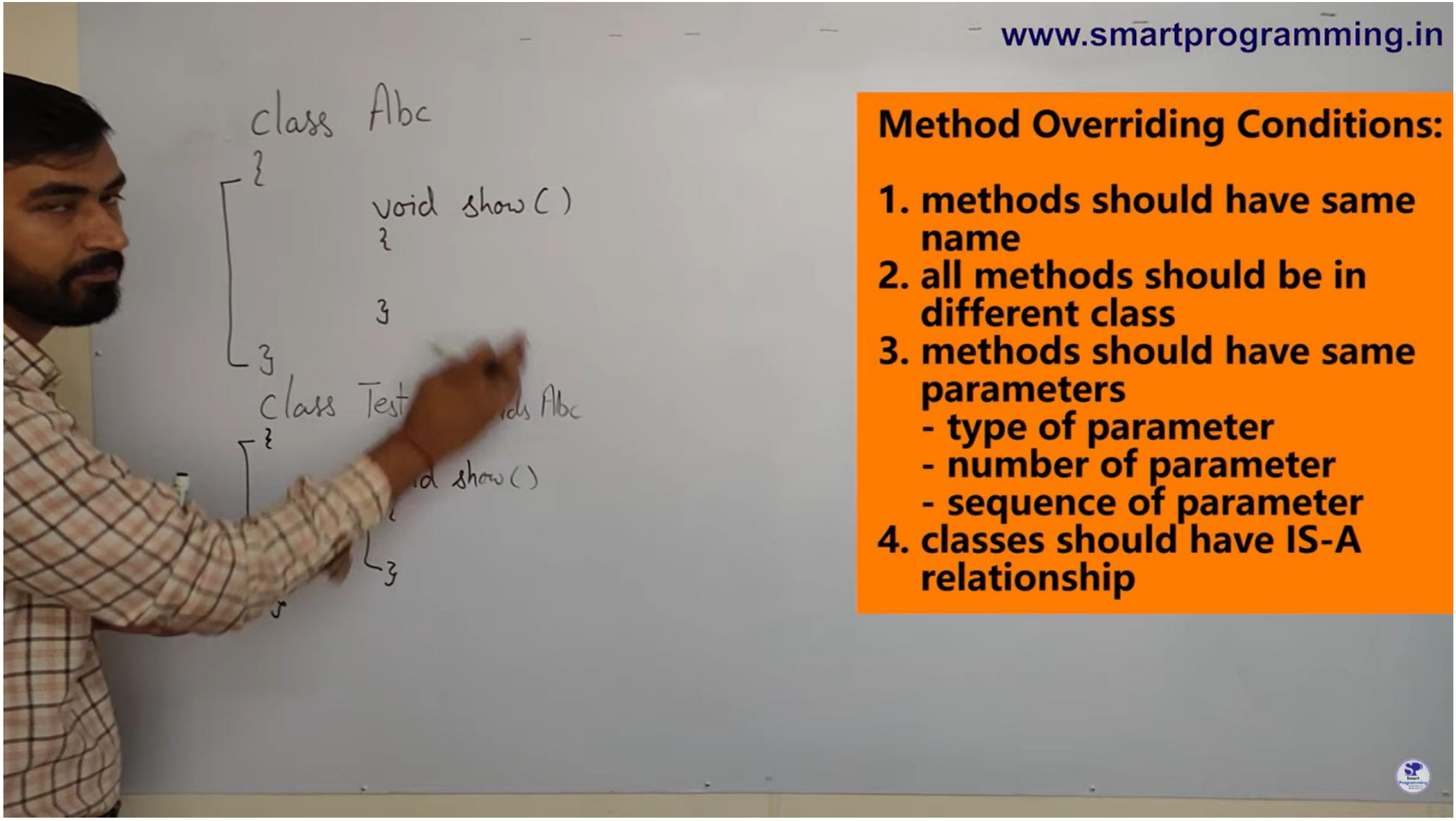
P1000




www.smartprogramming.in
final is the keyword which is used with class, method and variables

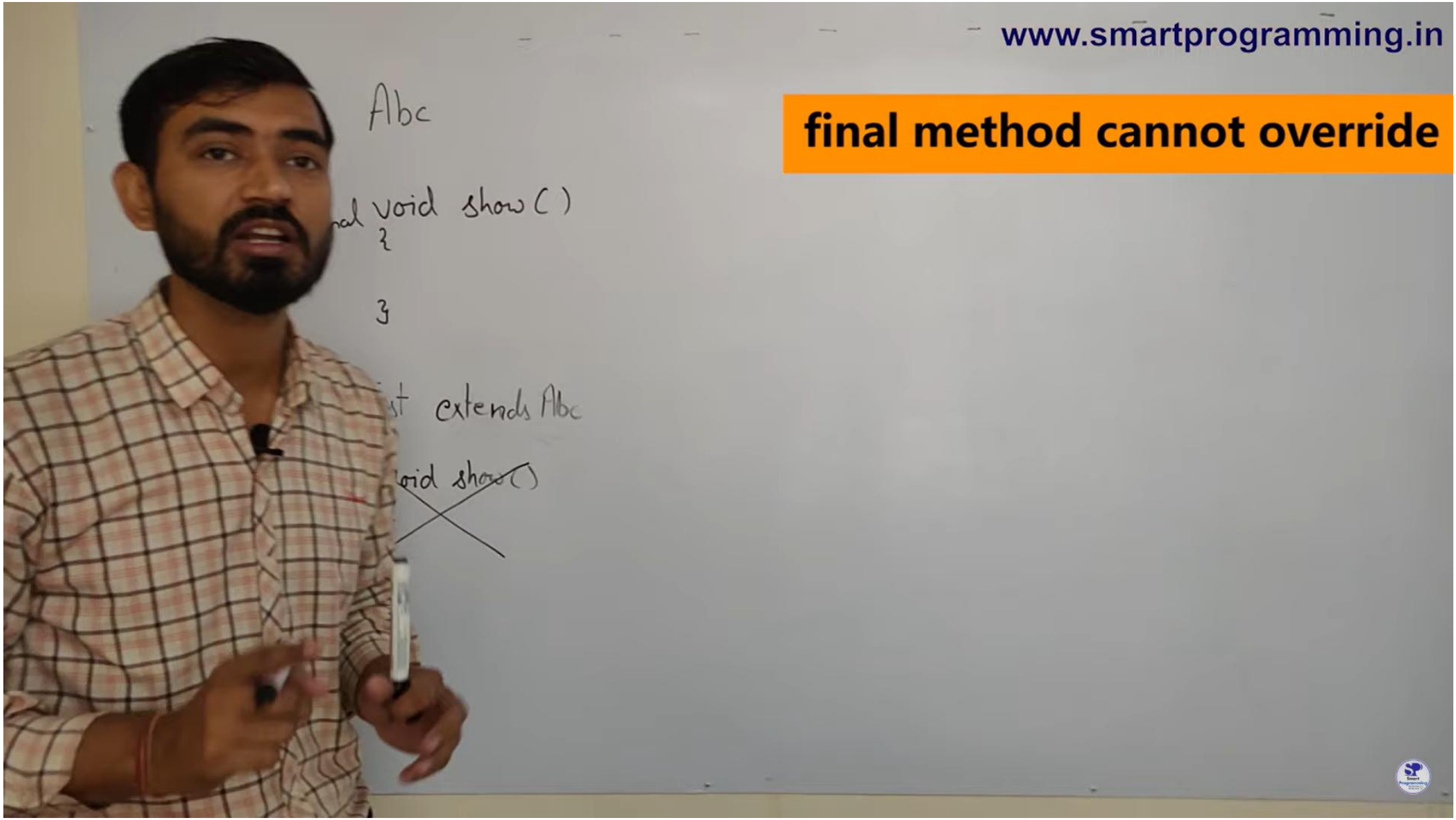


final class cannot be extend



Method Overriding Conditions:

1. methods should have same name
2. all methods should be in different class
3. methods should have same parameters
   - type of parameter
   - number of parameter
   - sequence of parameter
4. classes should have IS-A relationship


www.smartprogramming.in
final method cannot override
Abc
void show()
{
}
extends Abc


www.smartprogramming.in
class Abc
{
final void
{
}
}
final int a=10;
the value of final variable
cannot be changed

Important Features of String Class

1. **String Constant Pool (SCP) :-** It is special memory location in heap area which stores String Literals.

2. **Immutable Objects :-** The String objects are immutable which means once String object is created its data or state can't be changed but a new string object is created.

3. **+ Operator for Strings :-** Multiple Strings can be concatinated using + operator.

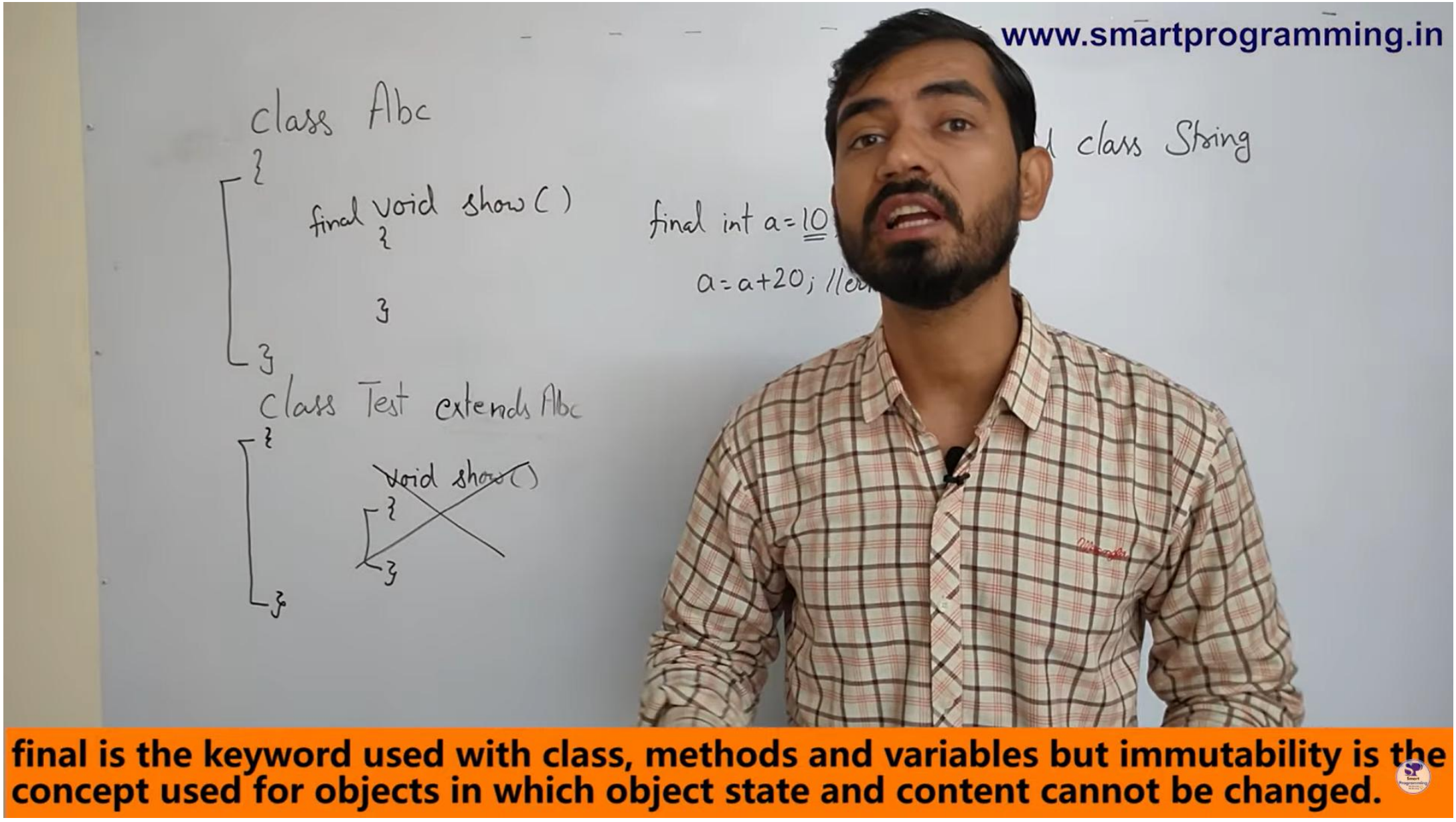
4. **Security:-** The parameters used for network connections, database connection URLs, usernames/ passwords, etc are represented in Strings. If it was mutable, these parameters could be changed easily.

5. **Class loading:-** String is used as an arguments for class loading. If mutable, it could result in the wrong class being loaded (because mutable objects change their state).

6. **Synchronization and Concurrency:-** Making String immutable automatically makes them thread safe thereby solving the synchronization issues.

7. **Memory management :-** When compiler optimizes our String objects, it seems that if two objects have the same value (a =" test", and b =" test") and thus we need only one string object (for both a and b, these two will point to the same object).

Why String class is final ?
String class has got special features which is not available to other java classes and making the String class final prevents subclasses that could break these assumptions

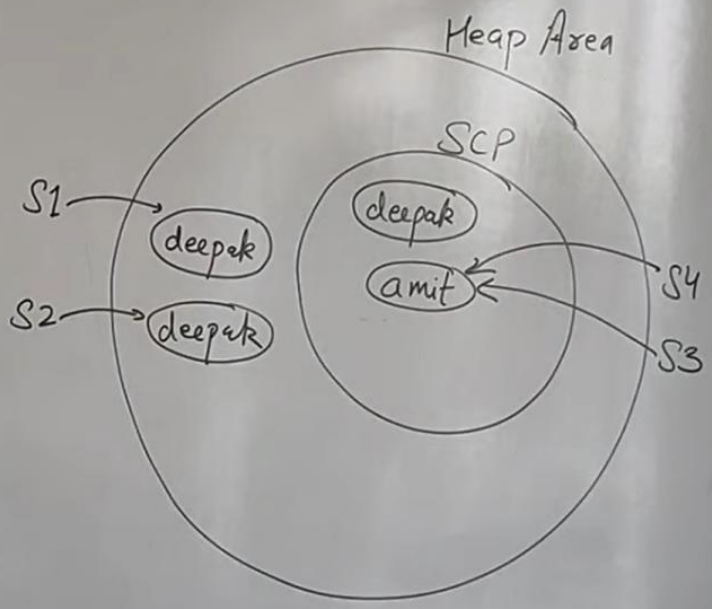


final is the keyword used with class, methods and variables but immutability is the concept used for objects in which object state and content cannot be changed.

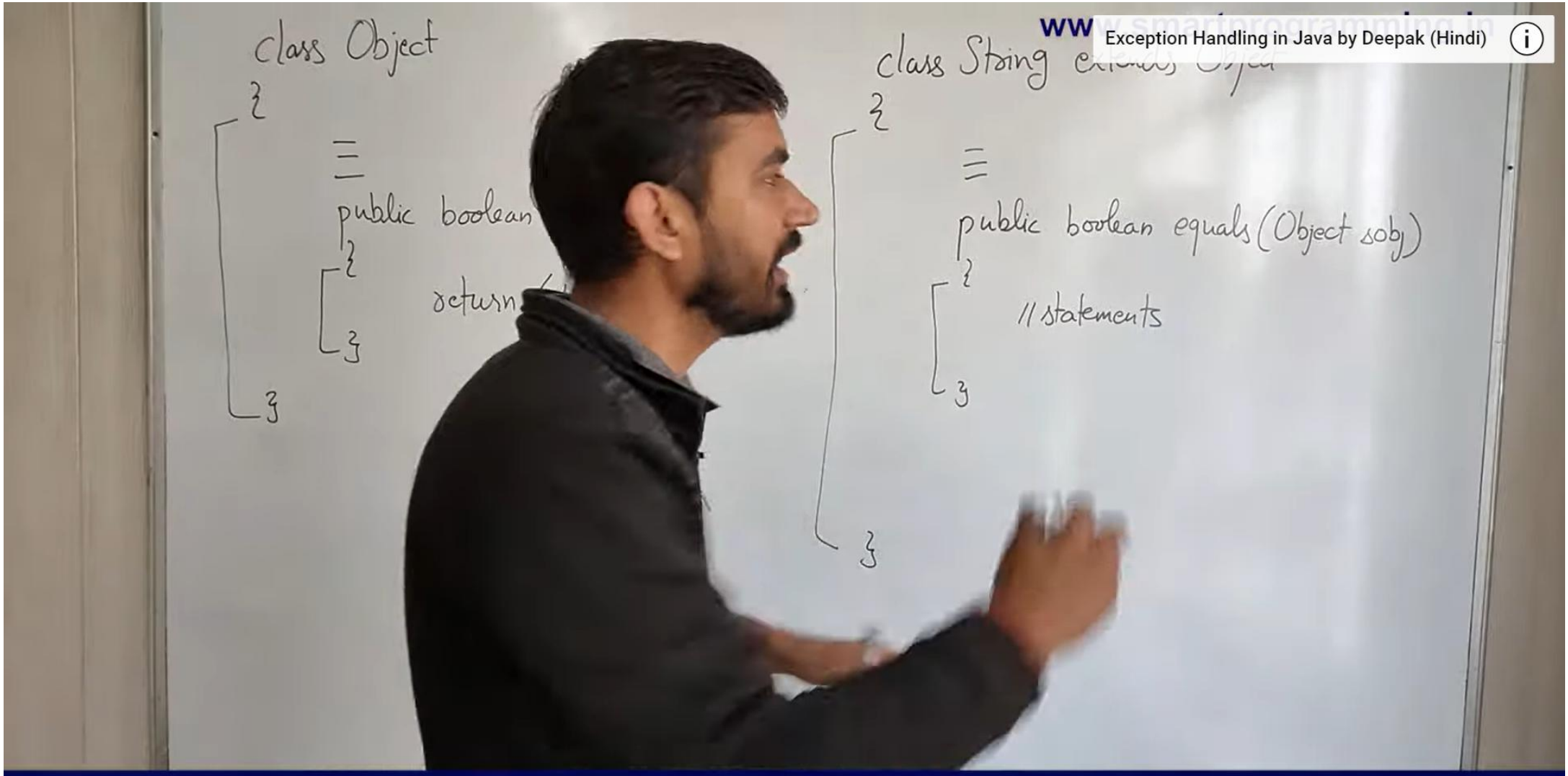
== operator is used for reference comparison (address comparison). It means == operator checks if both objects point to the same memory location or not.

.equals() method is used for content comparison (in String class). It means .equals() method is used to check object value.

```
String s1 = new String("deepak");
String s2 = new String("deepak");
S.o.p(s1==s2); // false
String s3 = "amit";
String s4 = "amit";
S.o.p(s3==s4); // true
```



== operator is used for reference comparison (address comparison). It means
== operator checks if both objects point to the same memory location or not.



.equals() method of String class is used for content comparison i.e it is used to check object value.

StringDemo - NetBeans IDE 8.0.2

www.smartprogramming.in

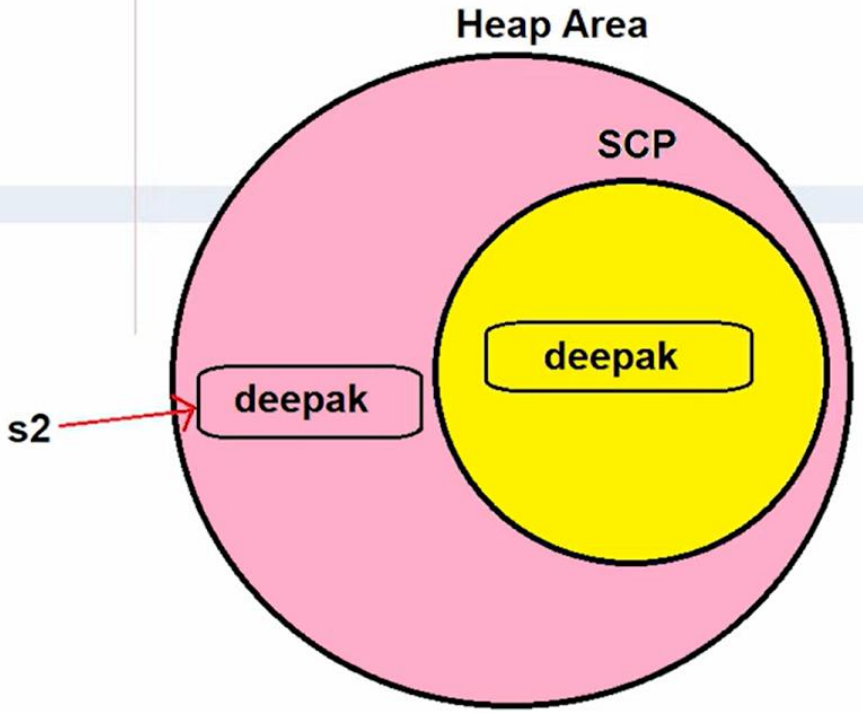
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

StringDemo.java | String.java

Source | History

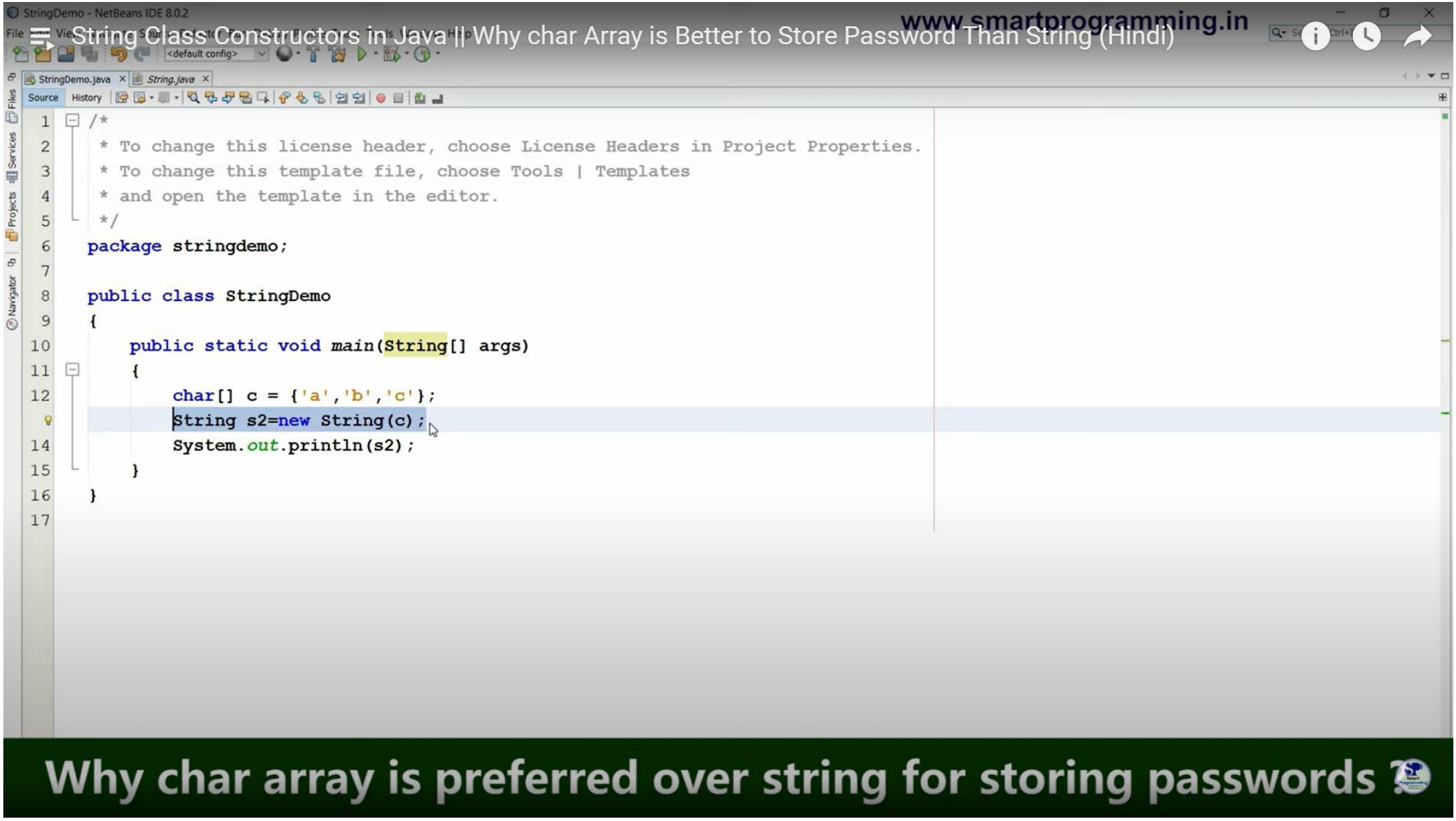
```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package stringdemo;
7
8  public class StringDemo
9  {
10     public static void main(String[] args)
11     {
12         String s2=new String("deepak");
13     }
14 }
15
```

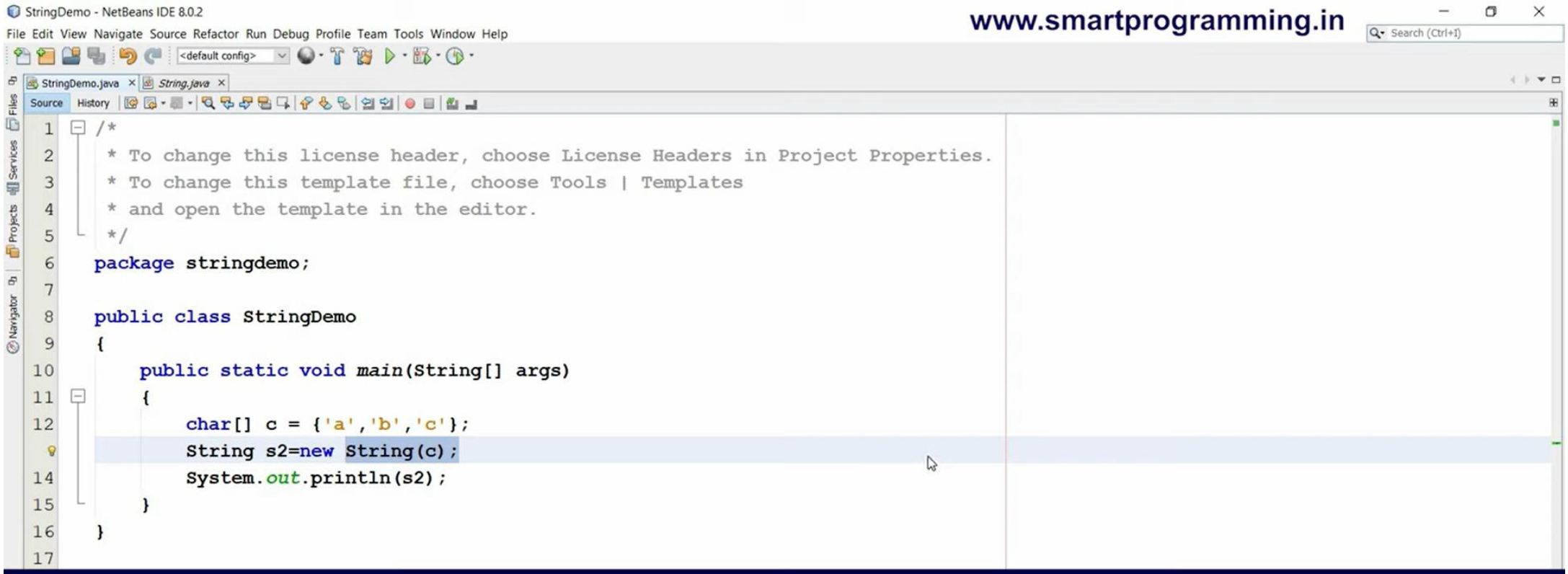


HTTP Server Monitor | Output

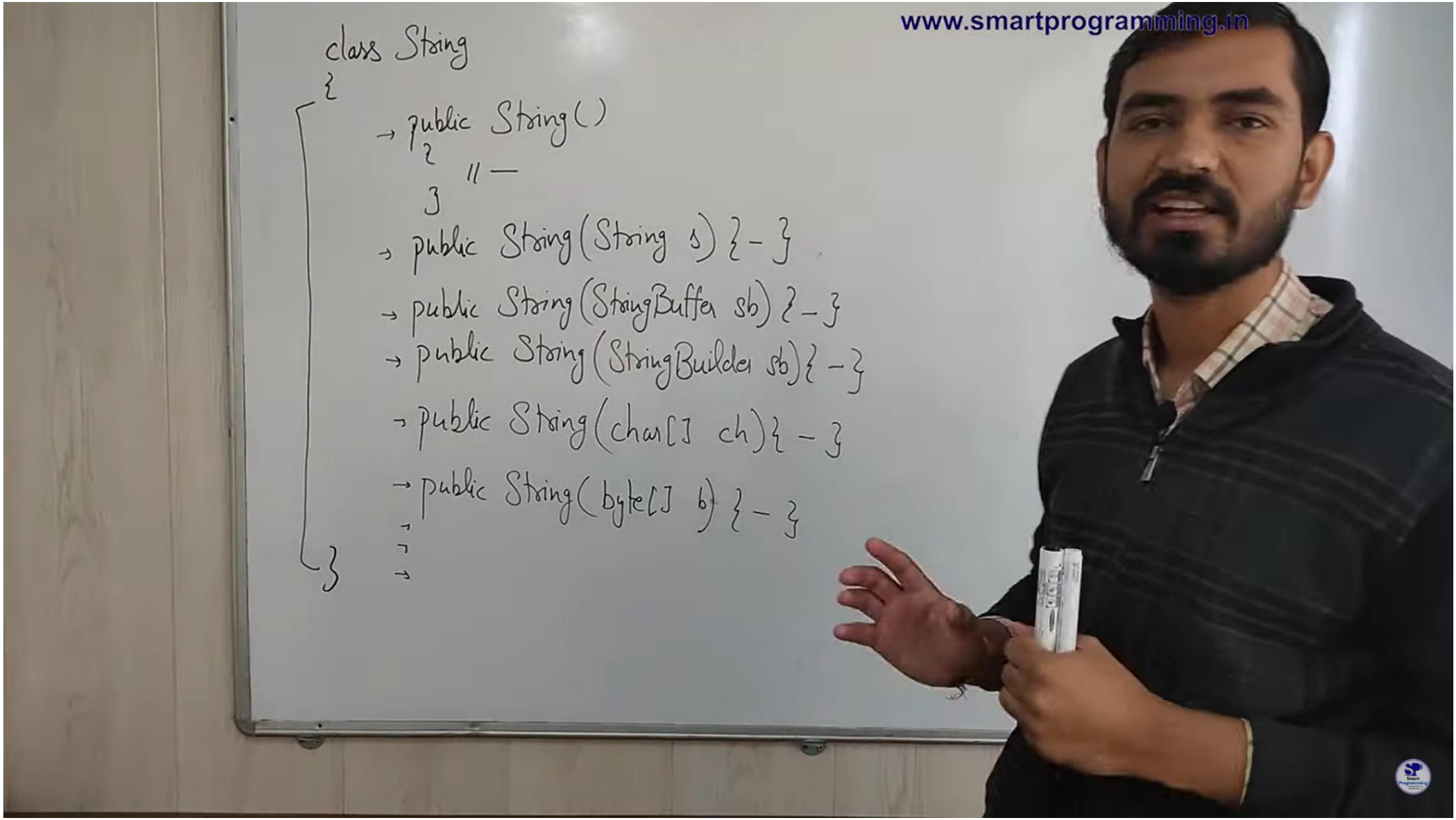
12:37 INS

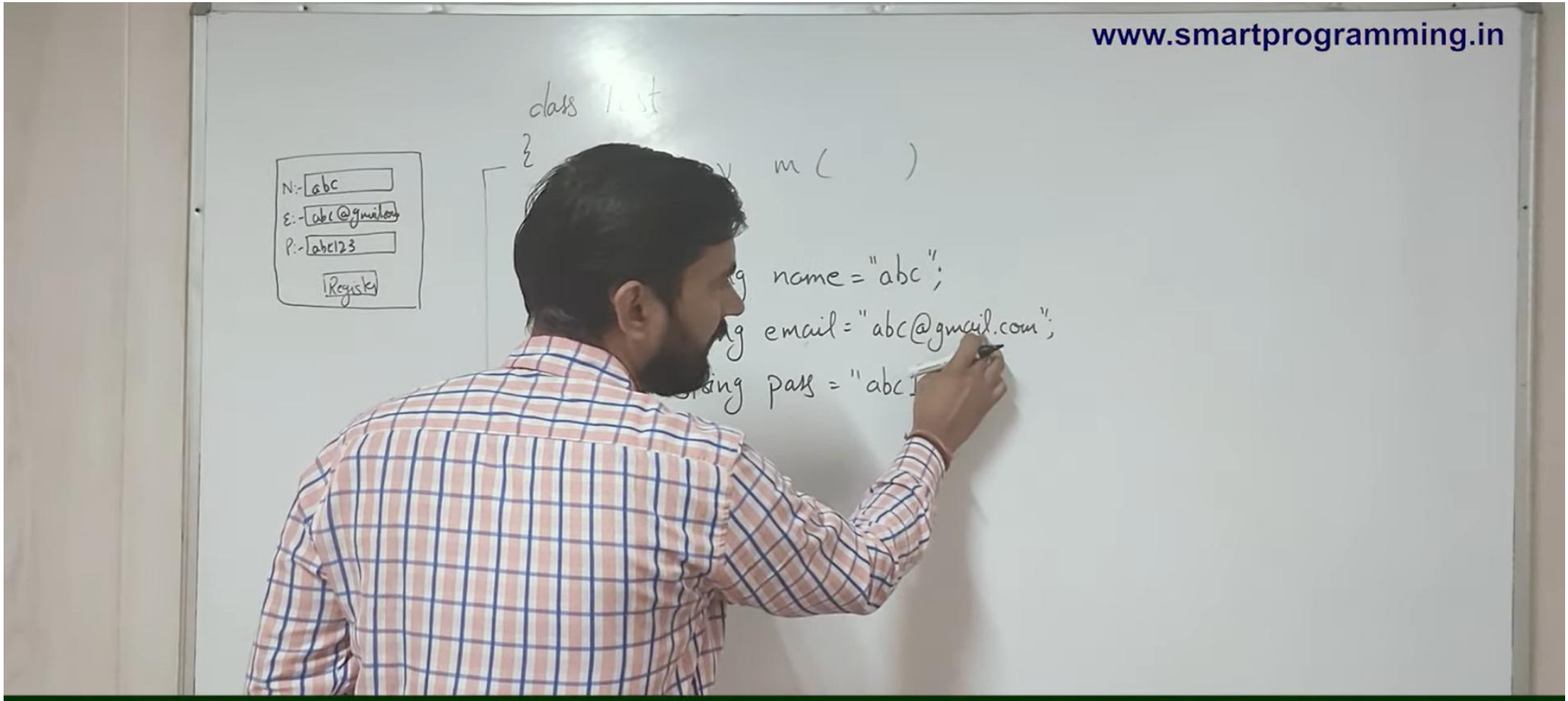
18% ENG 09:16 AM 11-12-2019


StringDemo - NetBeans IDE 8.0.2
String Class Constructors in Java || Why char Array is Better to Store Password Than String (Hindi)
www.smartprogramming.in
StringDemo.java
String.java
Source
History
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package stringdemo;
public class StringDemo
{
public static void main(String[] args)
{
char[] c = {'a','b','c'};
String s2=new String(c);
System.out.println(s2);
}
}
Why char array is preferred over string for storing passwords ?

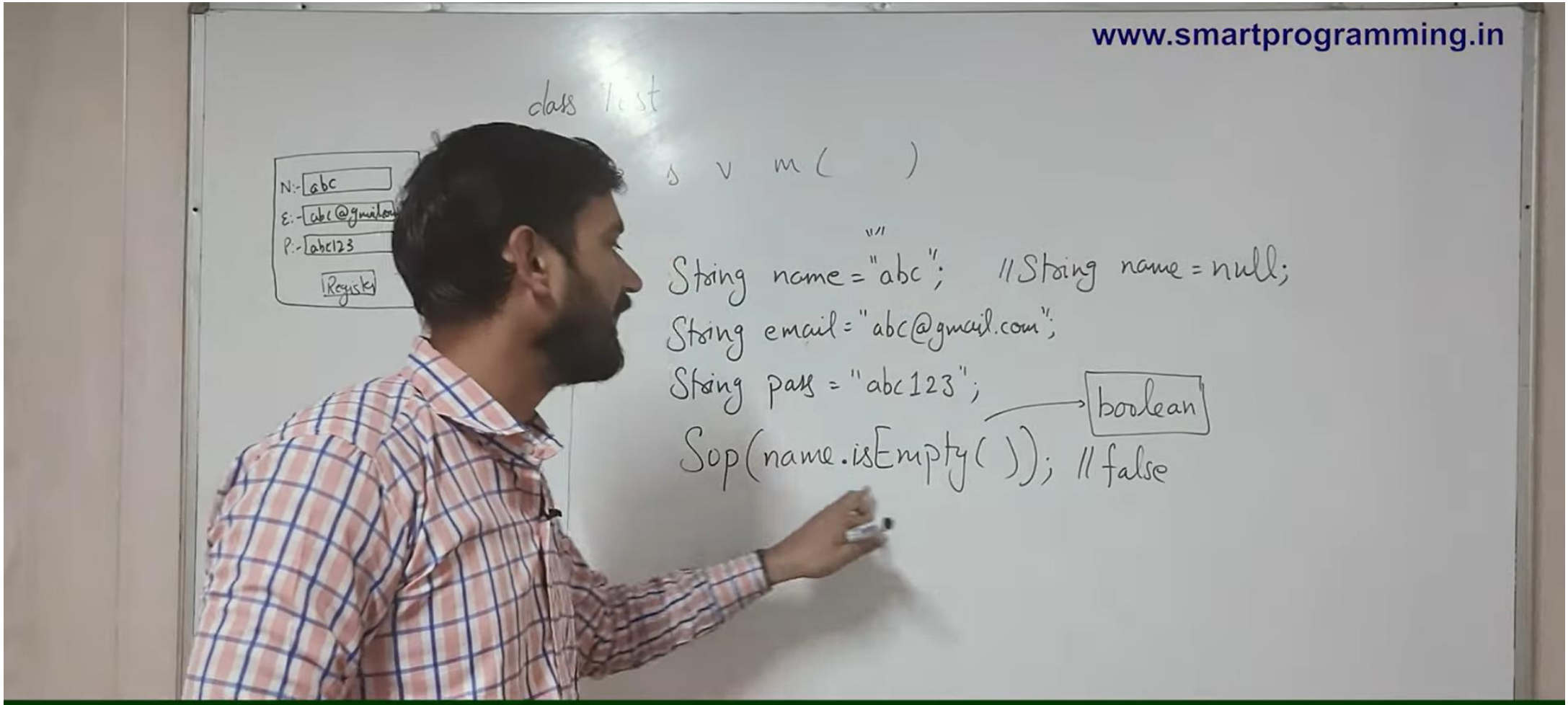


1. String objects are immutable in Java and therefore if a password is stored as plain text it will be available in memory until Garbage collector clears it, but String objects are stored in String Literal pool for re-usability and garbage collection is not applicable in SCP, which is a security threat.
With an array, you can explicitly wipe the data after you're done with it. You can overwrite the array with anything you like, and the password won't be present anywhere in the system, even before garbage collection.

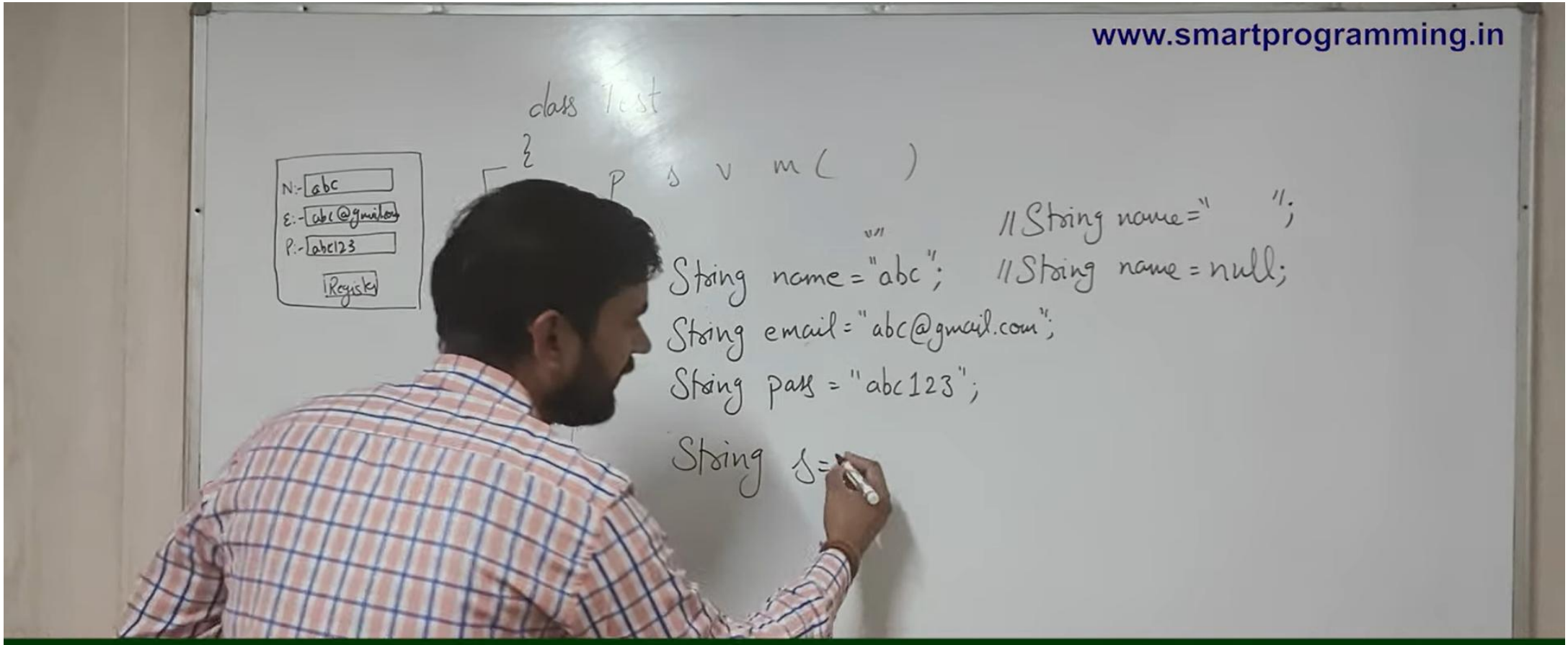

www.smartprogramming.in
class String
{
→ public String()
{
// —
}
→ public String(String s) {-}
→ public String(StringBuffer sb) {-}
→ public String(StringBuilder sb) {-}
→ public String(char[] ch) {-}
→ public String(byte[] b) {-}
→
→
→
}



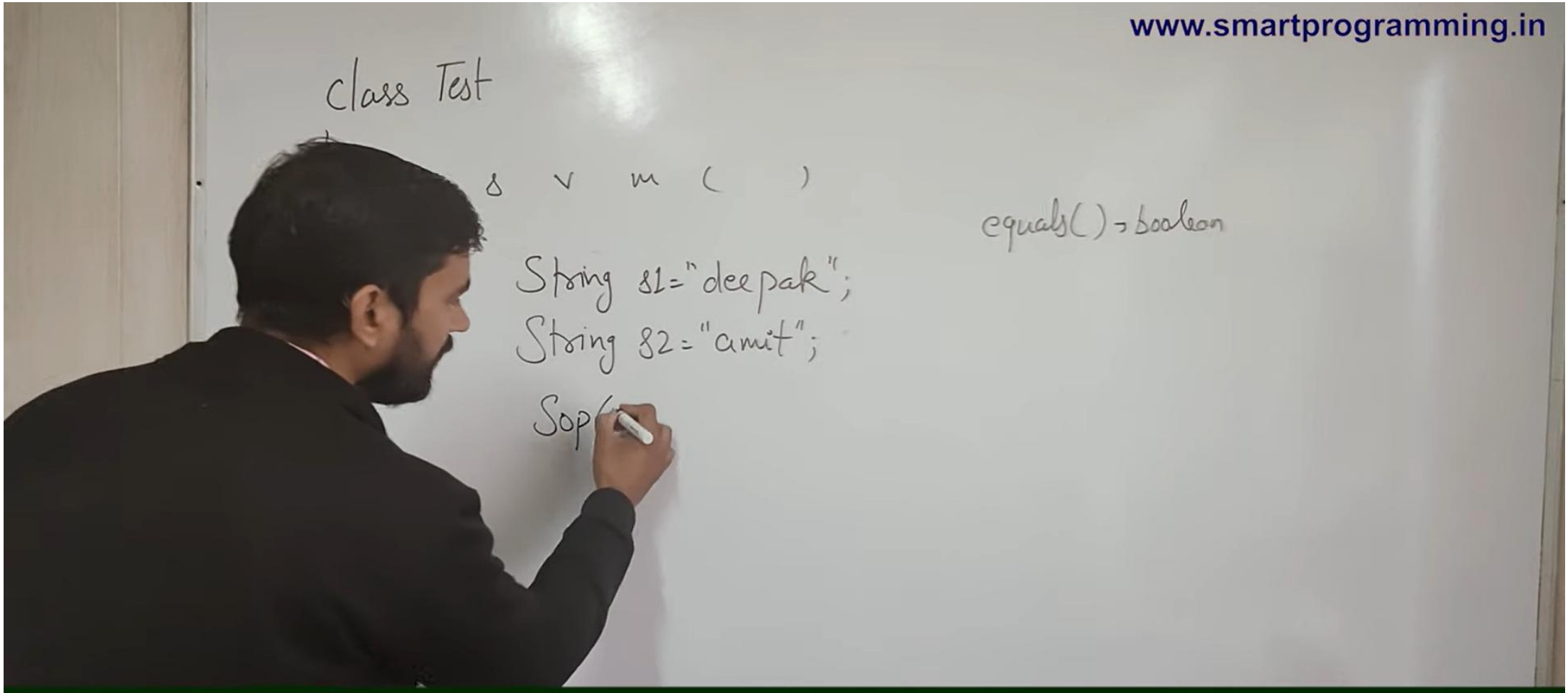
The string length() method counts the number of characters in the String and returns it in integer. This method returns the length of any string which is equal to the number of 16-bit Unicode characters in the string.



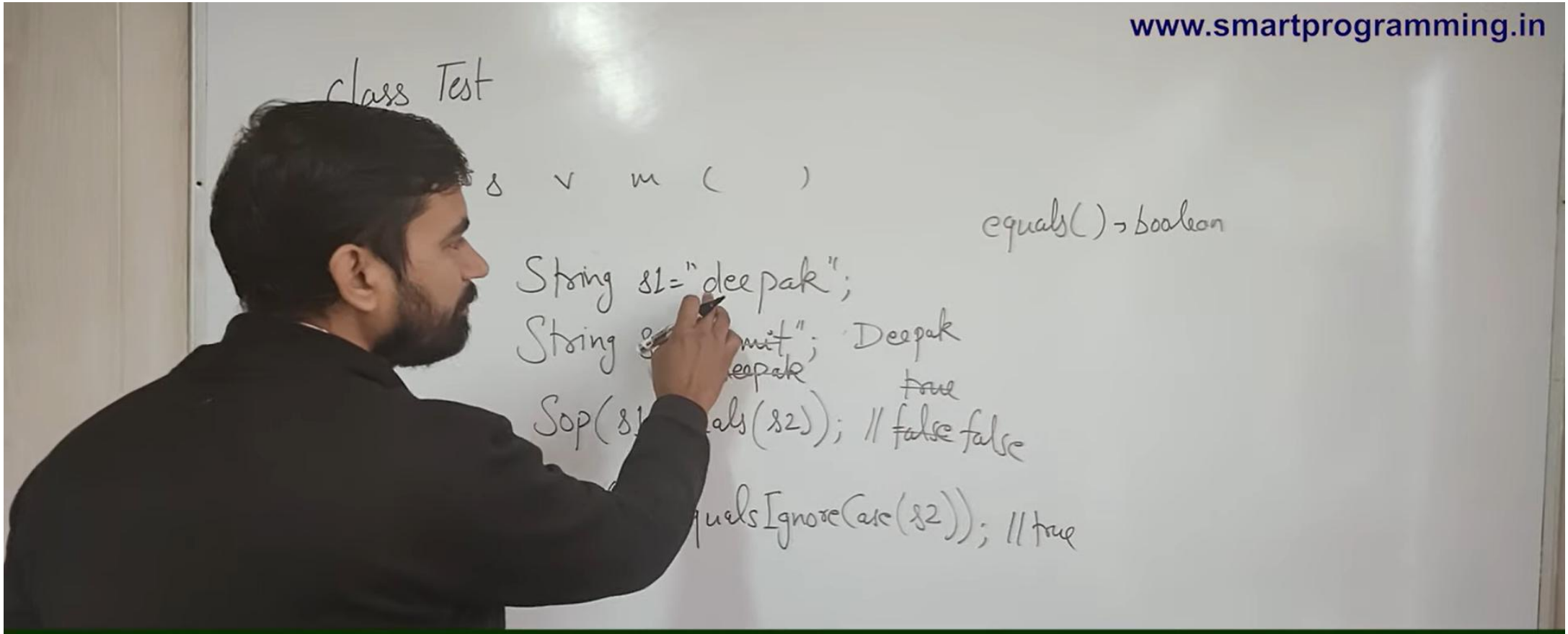
The isEmpty() method of String class is included in java string since JDK 1.6. This method returns true if the given string is empty, else it returns false, or can say that this method returns true if the length of the string is 0.



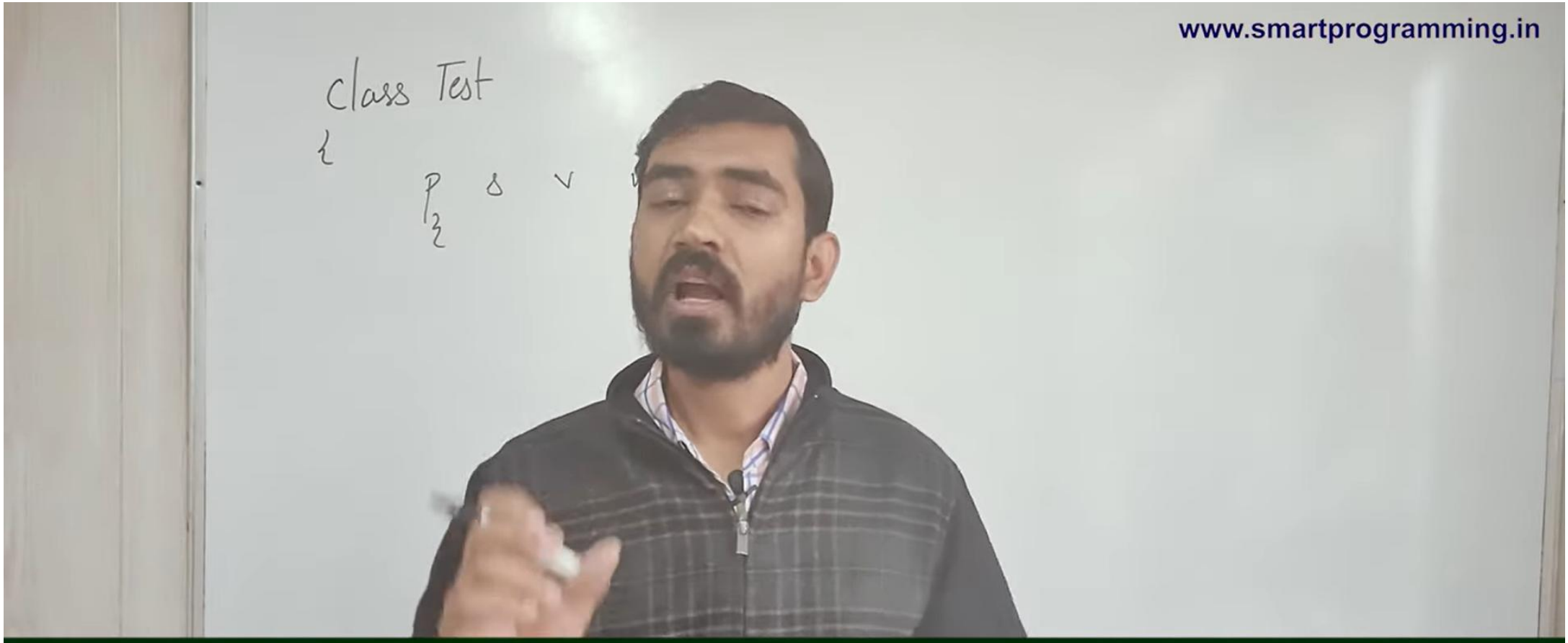
trim() method of String class eliminates only leading and trailing spaces. The Unicode value of space character is '\u0020'. The trim() method in java checks this Unicode value before and after the string, if it exists then removes the spaces and returns the omitted string.



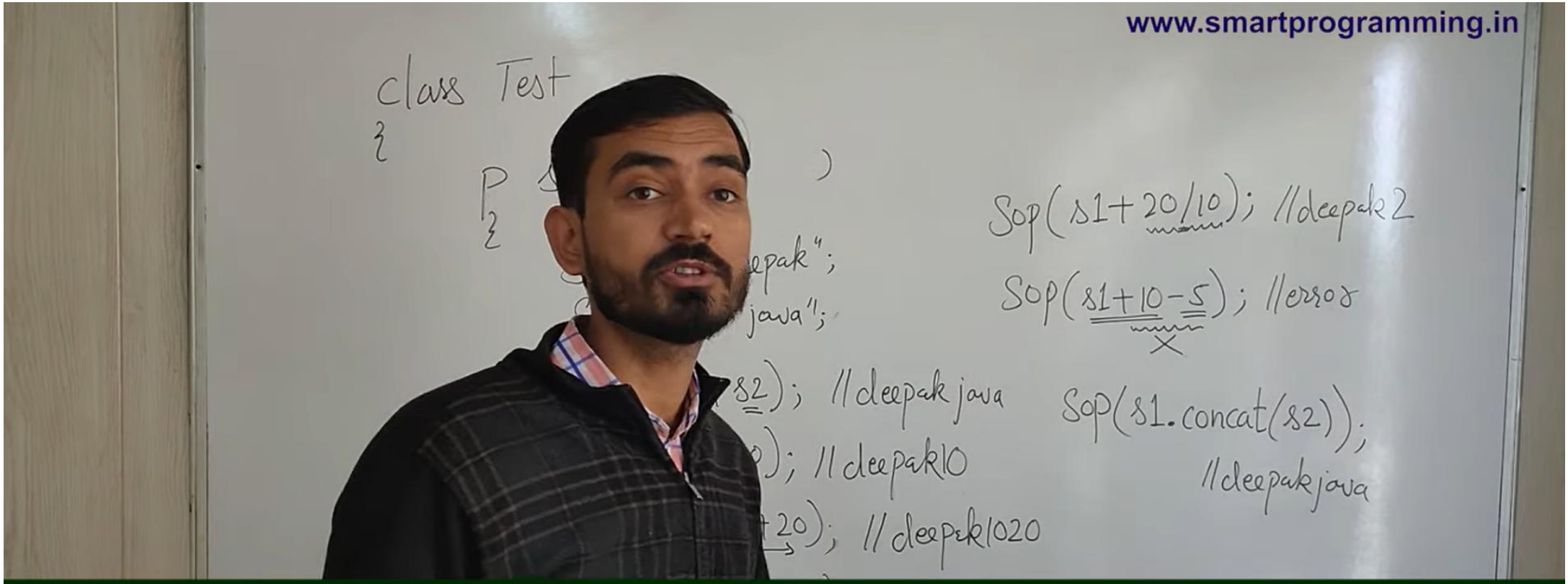
The equals() method compares the content of given two strings. If any character is not matched, it returns false. If all characters are matched,it returns true. (equals() and == is different which was explained in previous video)



The equalsIgnoreCase() Method is used to compare a specified String to another String, ignoring case considerations i.e. lower case or upper case. Two strings are considered equal ignoring case if they are of the same length and corresponding characters in the two strings are equal ignoring case.

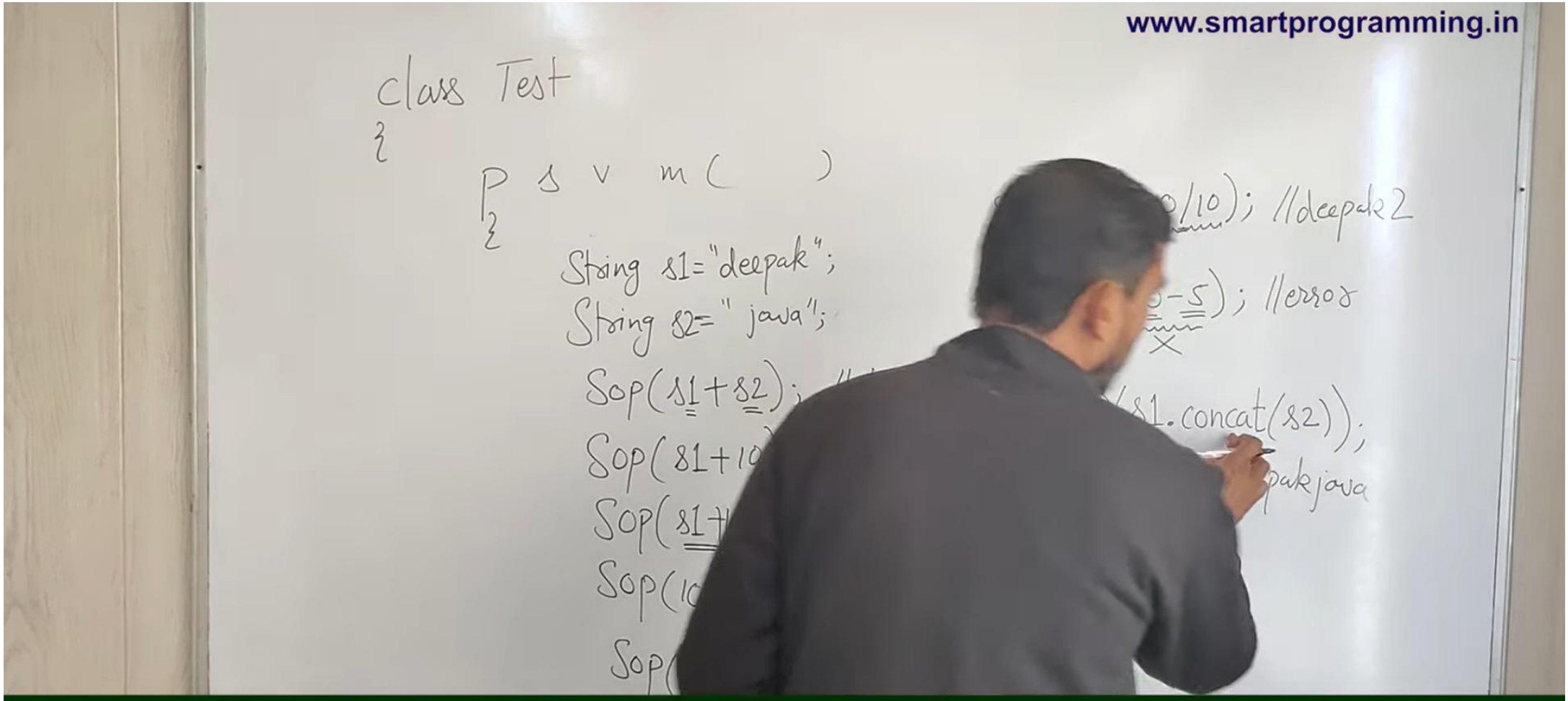


compareTo() and compareToIgnoreCase() method is used for comparing two strings lexicographically. Each character of both the strings is converted into a Unicode value for comparison. If both the strings are equal then this method returns 0 else it returns positive or negative value.

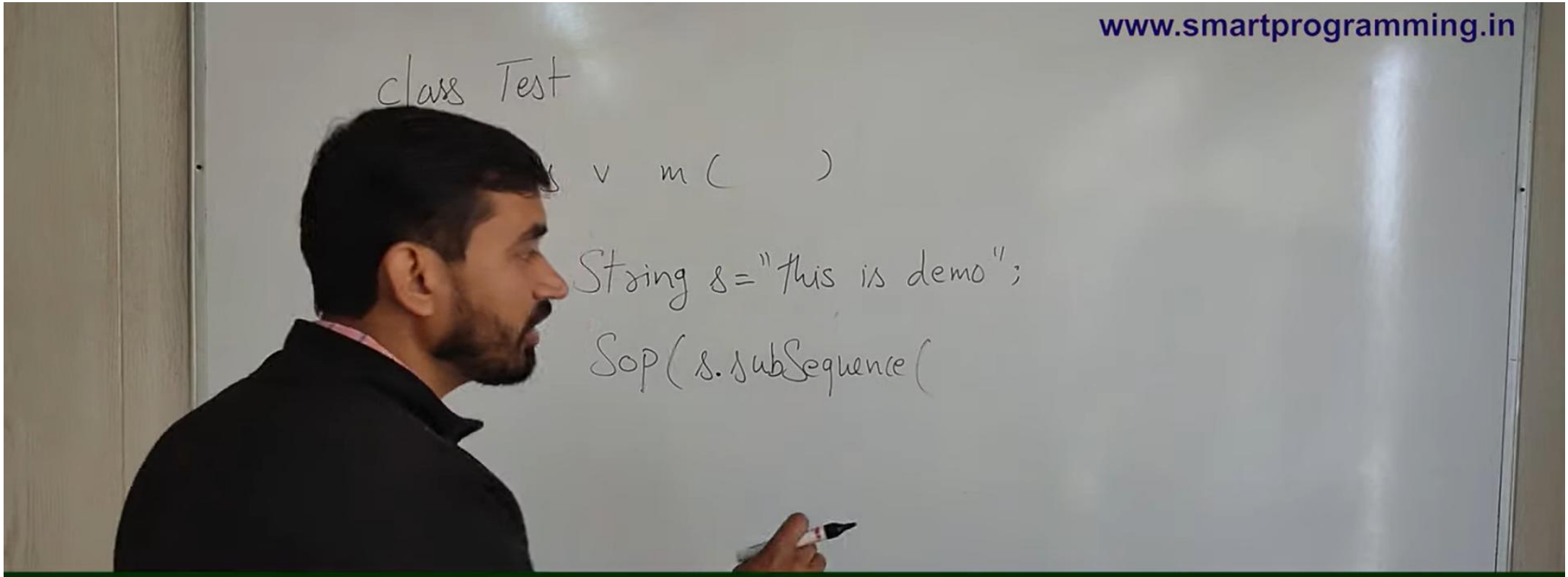


join(CharSequence delimiter, CharSequence... elements) is the static method which concatenates the given elements with the delimiter and returns the concatenated string. It was included in JDK 1.8 version.

Note that if an element is null, then null is added and if delimiter is null then it will throw "java.lang.NullPointerException"

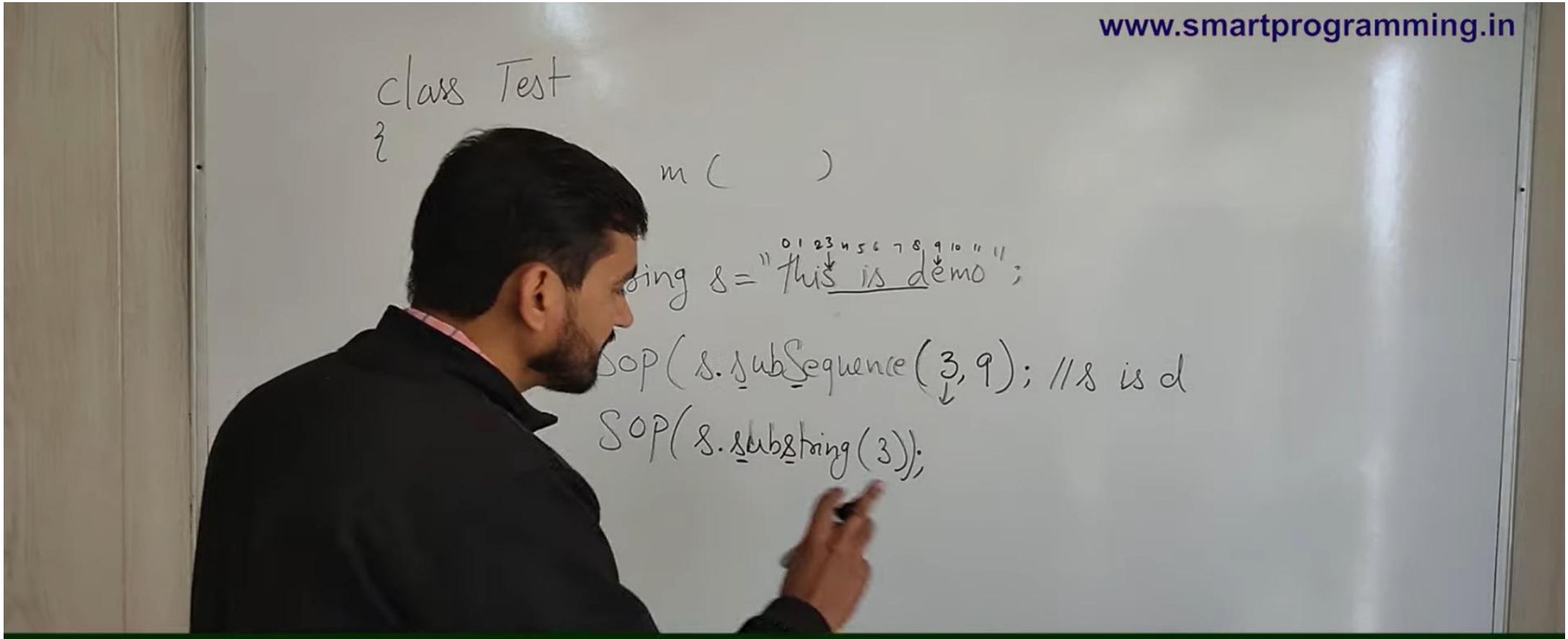


concat(String str) method concatenates one string to the end of another string. This method returns a string with the value of the string passed into the method, appended to the end of the string.

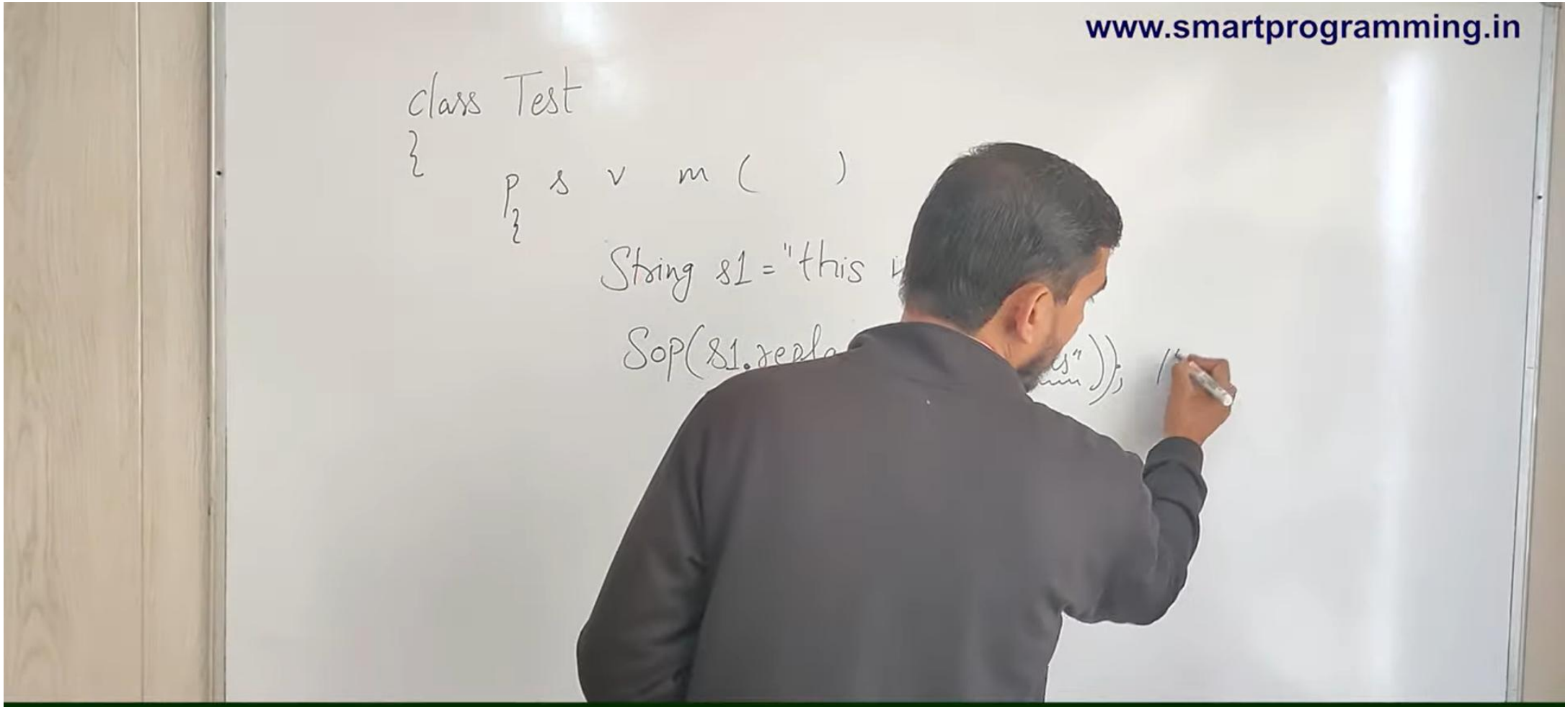


The subSequence(int beginIndex, int endIndex) method returns a CharSequence. The subsequence starts with the char value at the specified index and ends with the char value at (end-1).

It throws java.lang.StringIndexOutOfBoundsException exception if any index position value is negative.

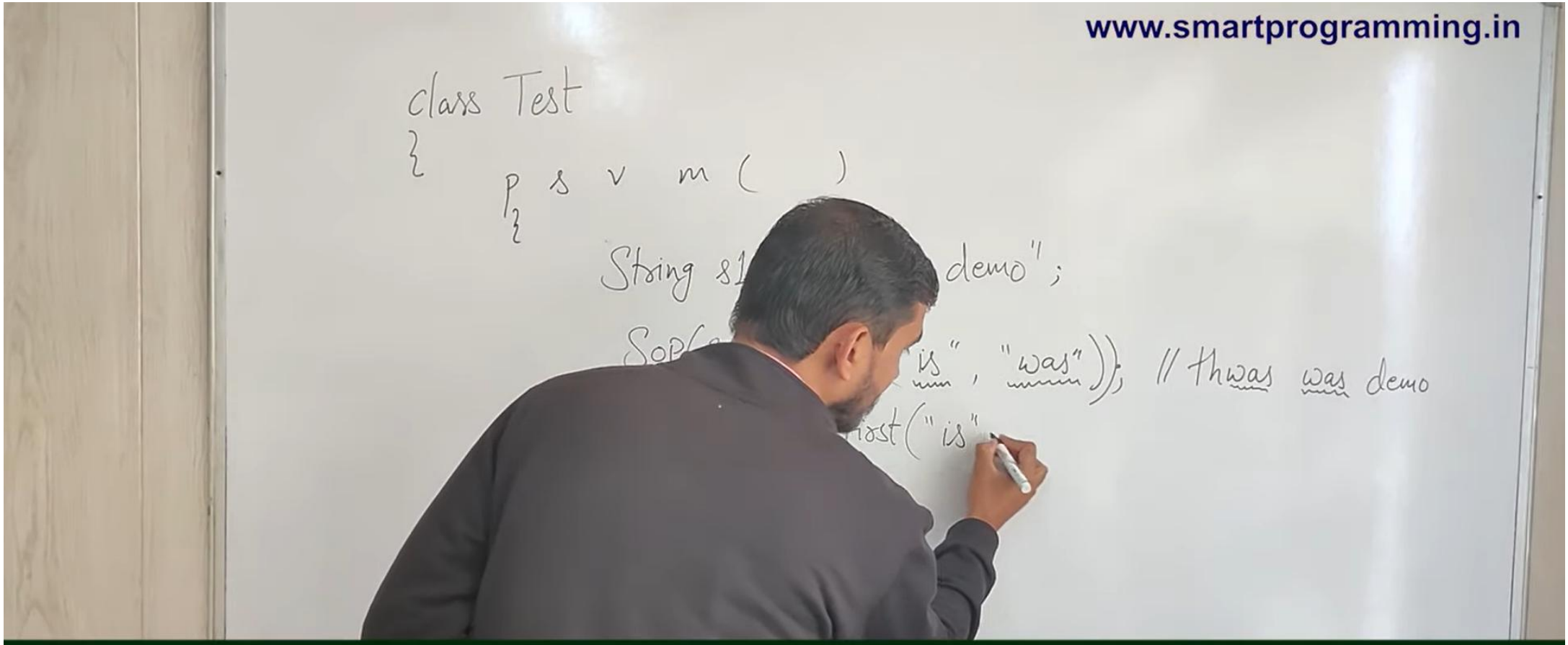


There are two substring methods of String class i.e. substring(int beginIndex) and substring(int beginIndex, int endIndex). It returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1.



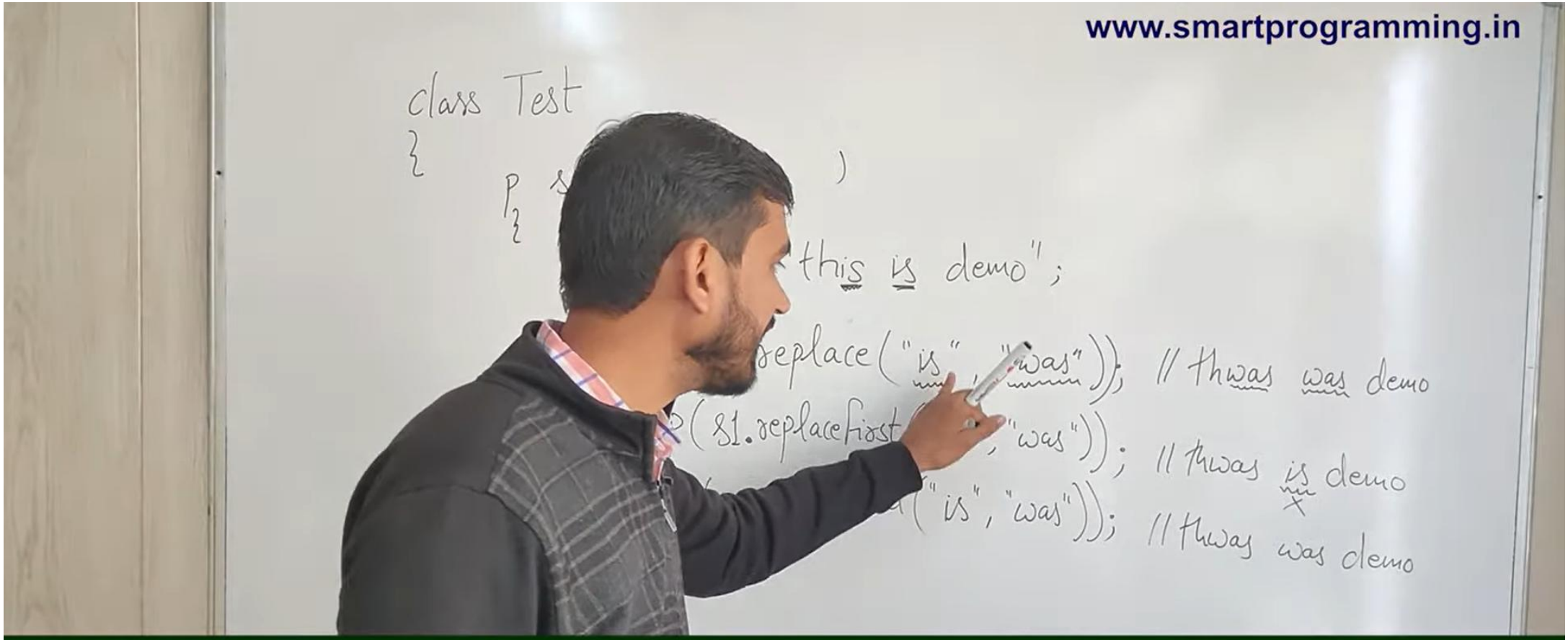
replace(char oldChar, char newChar) method returns a string replacing all the old characters or CharSequence to new chararacters or CharSequence.

This method was introduced in JDK 1.5 version.

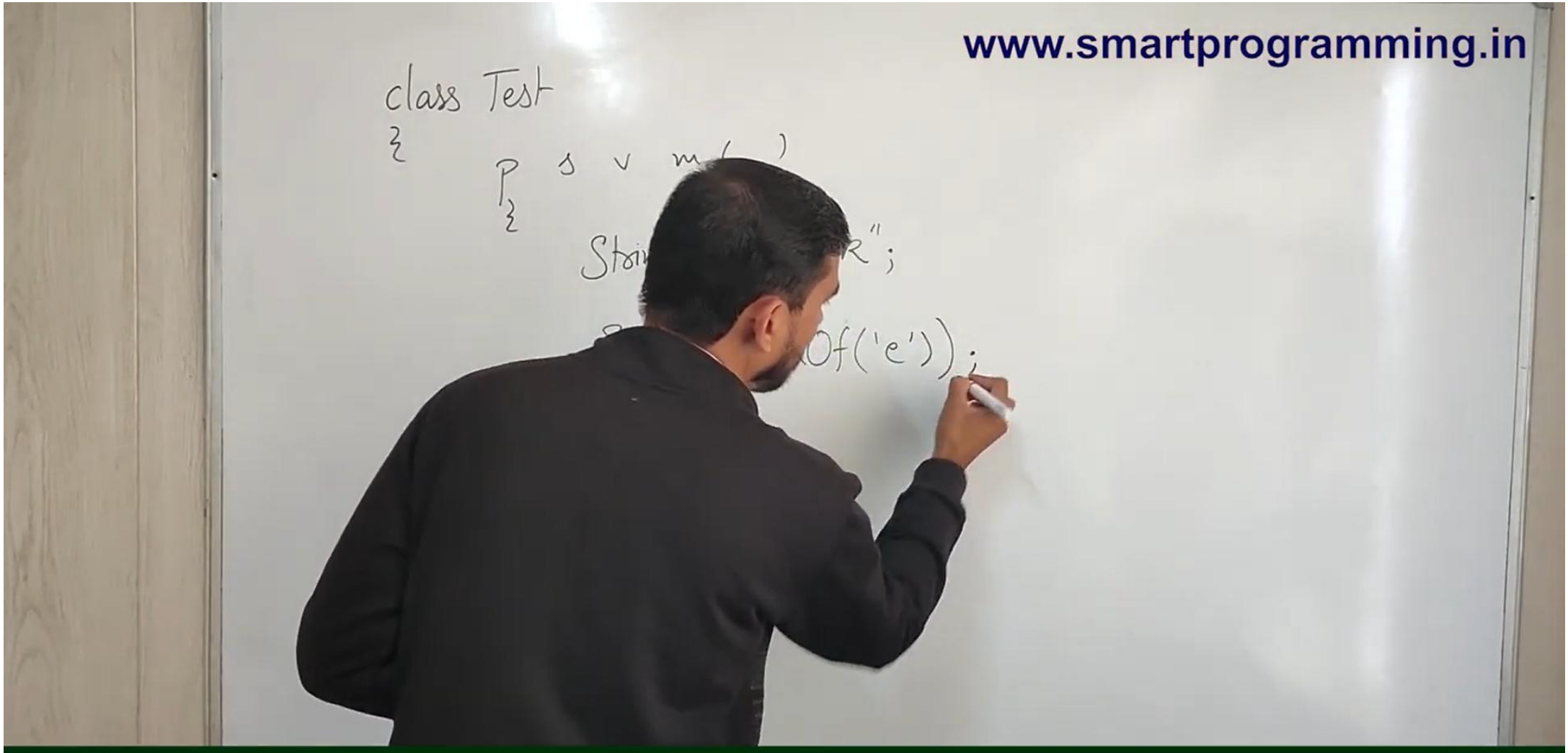


replaceFirst(String regex, String replacement) method replaces the first substring that fits the specified regular expression with the replacement String.

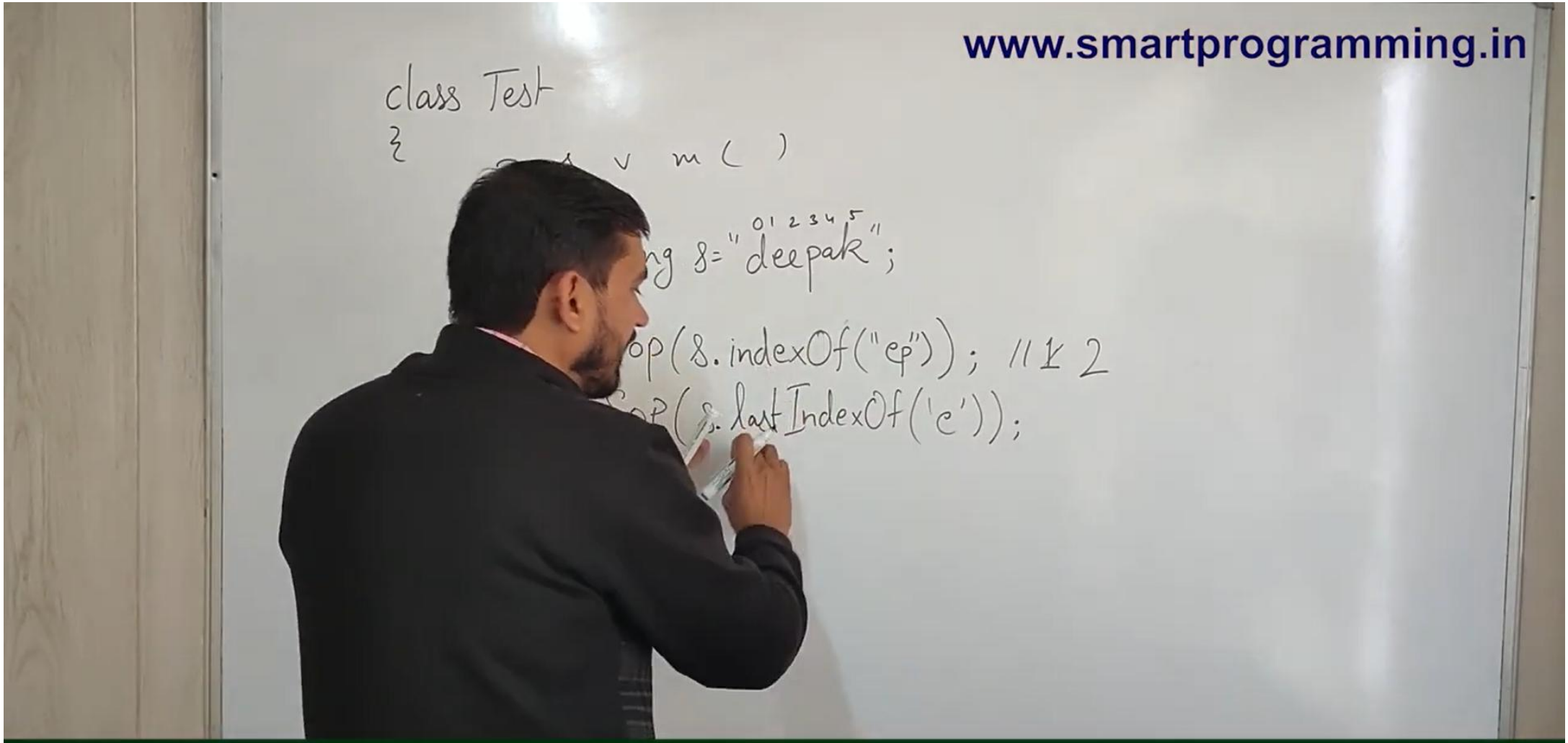
If the specified regular expression(regex) is not valid, then it will provide "java.util.regex.PatternSyntaxException" exception



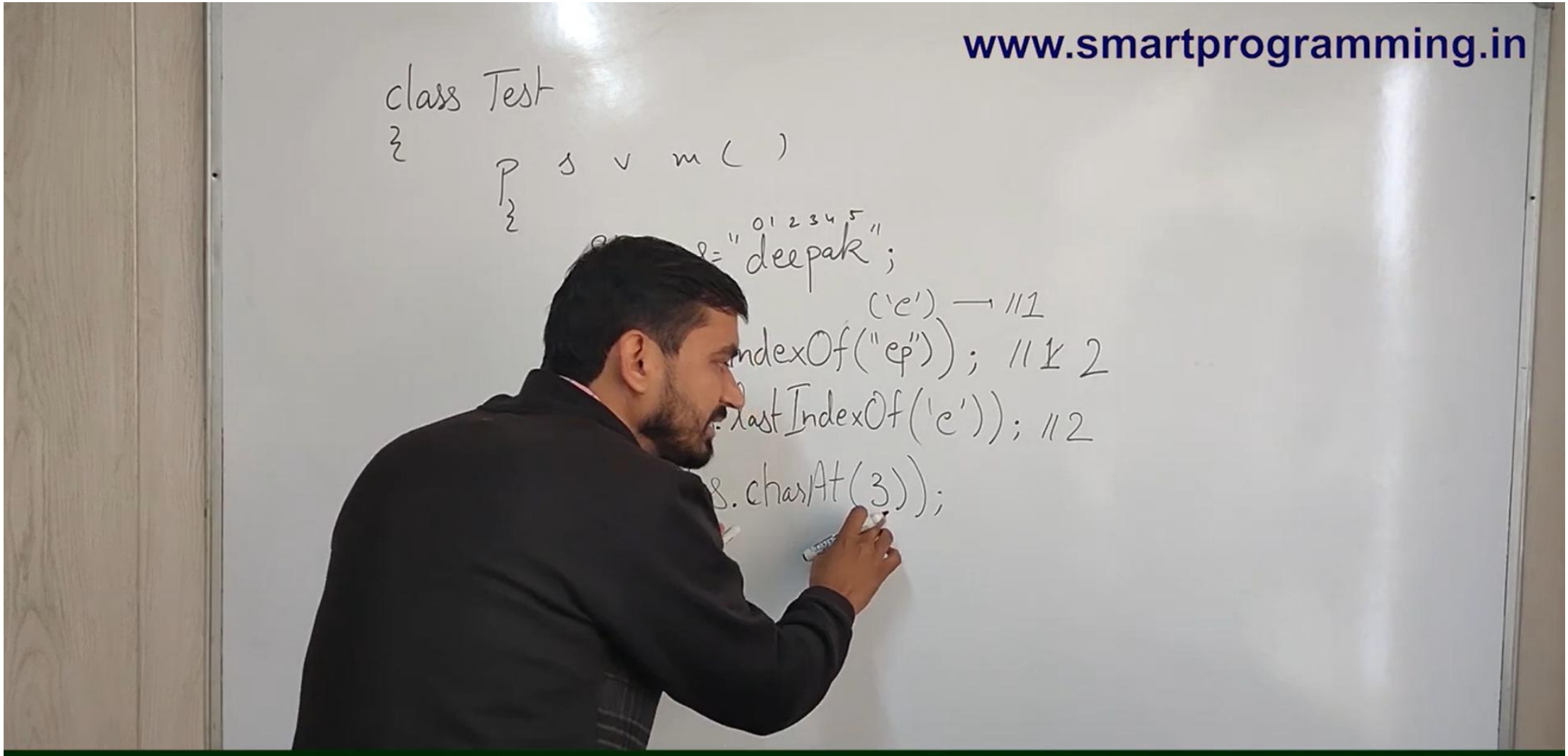
replaceAll(String regex, String replacement) method replaces all the substrings that fits the specified regular expression with the replacement String.
If the specified regular expression(regex) is not valid, then it will provide "java.util.regex.PatternSyntaxException" exception



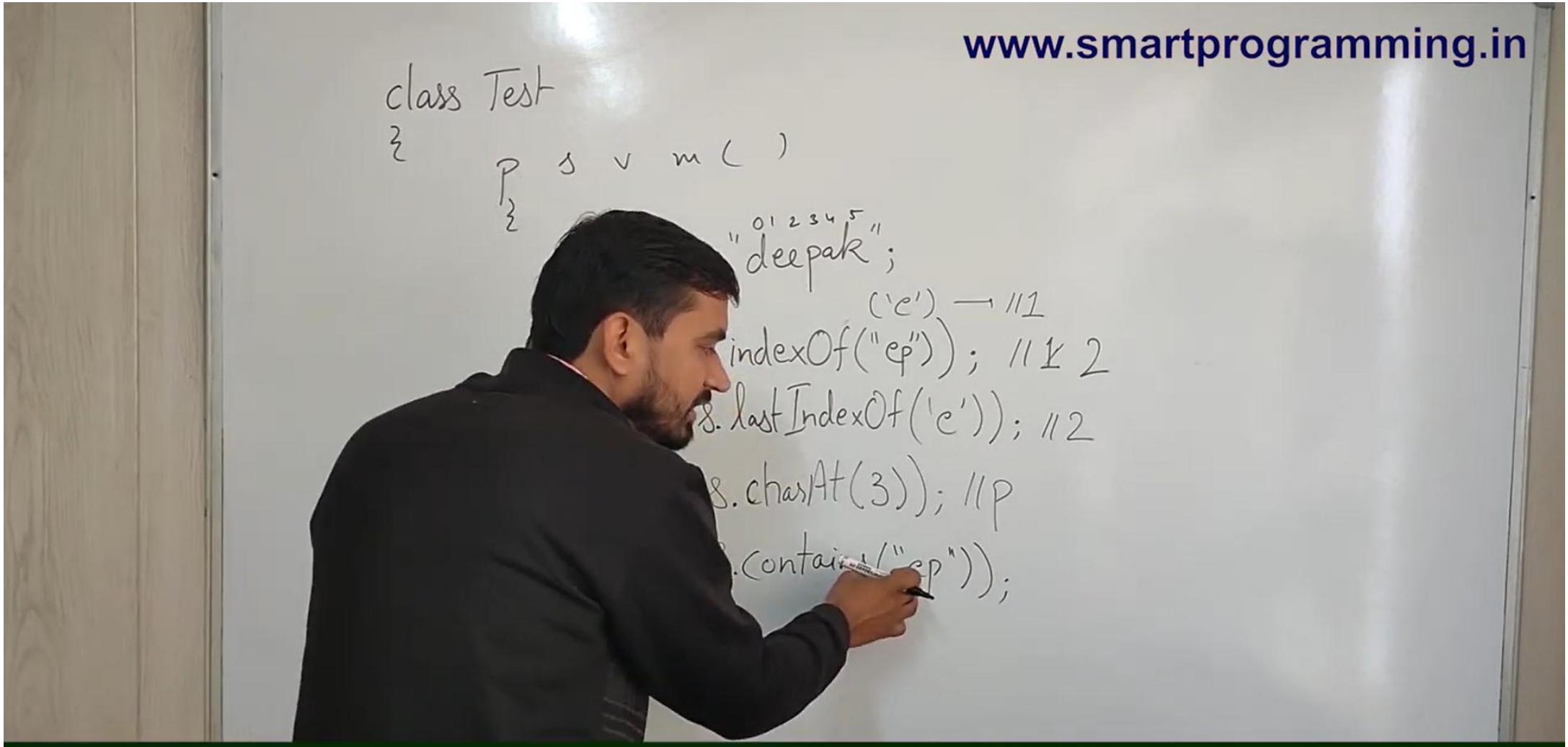
The indexOf() method returns the position of the first occurrence of specified character(s) in a string or return -1, if the character does not occur.



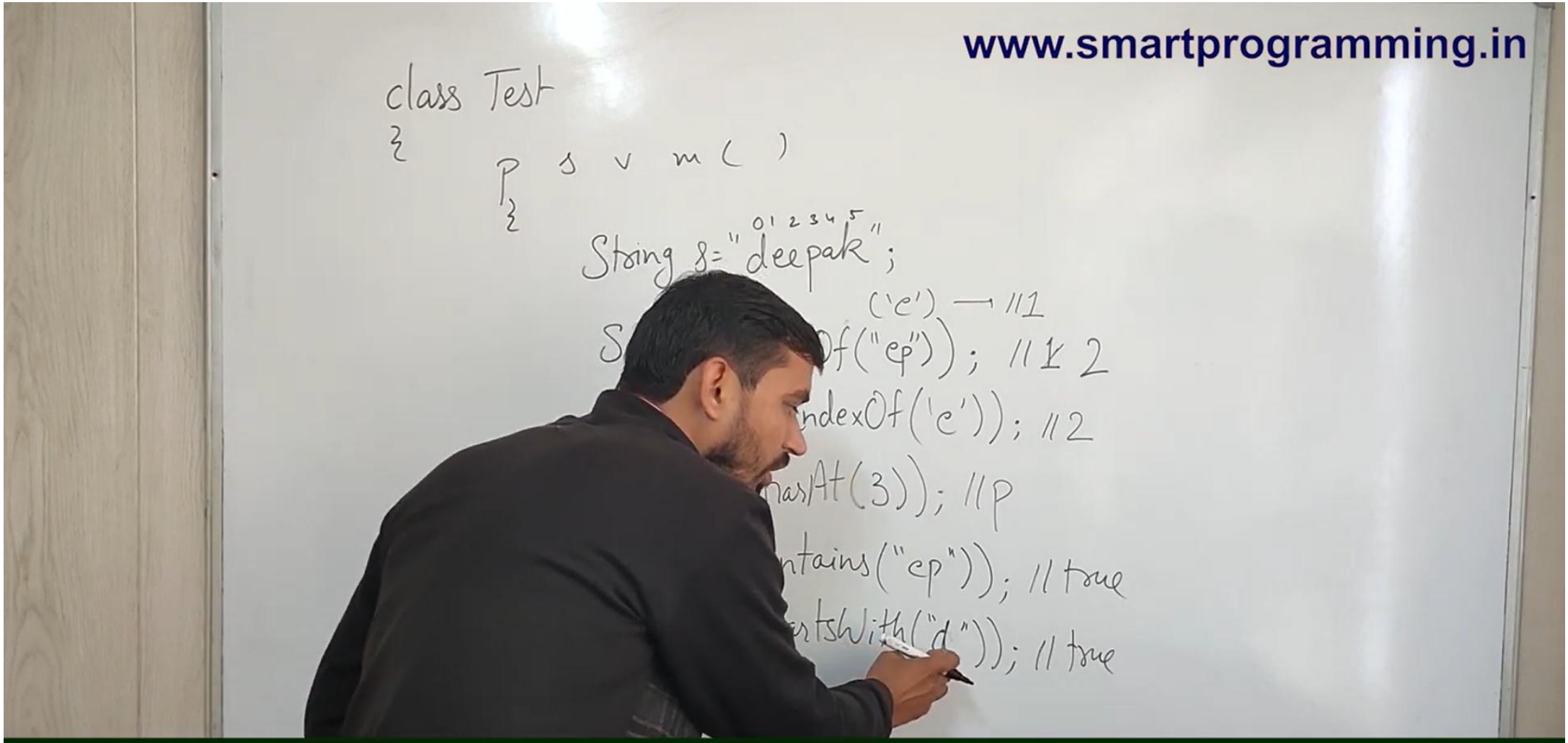
The lastIndexOf() method returns the position of the last occurrence of the specified character(s) in a string or return -1, if the character does not occur.



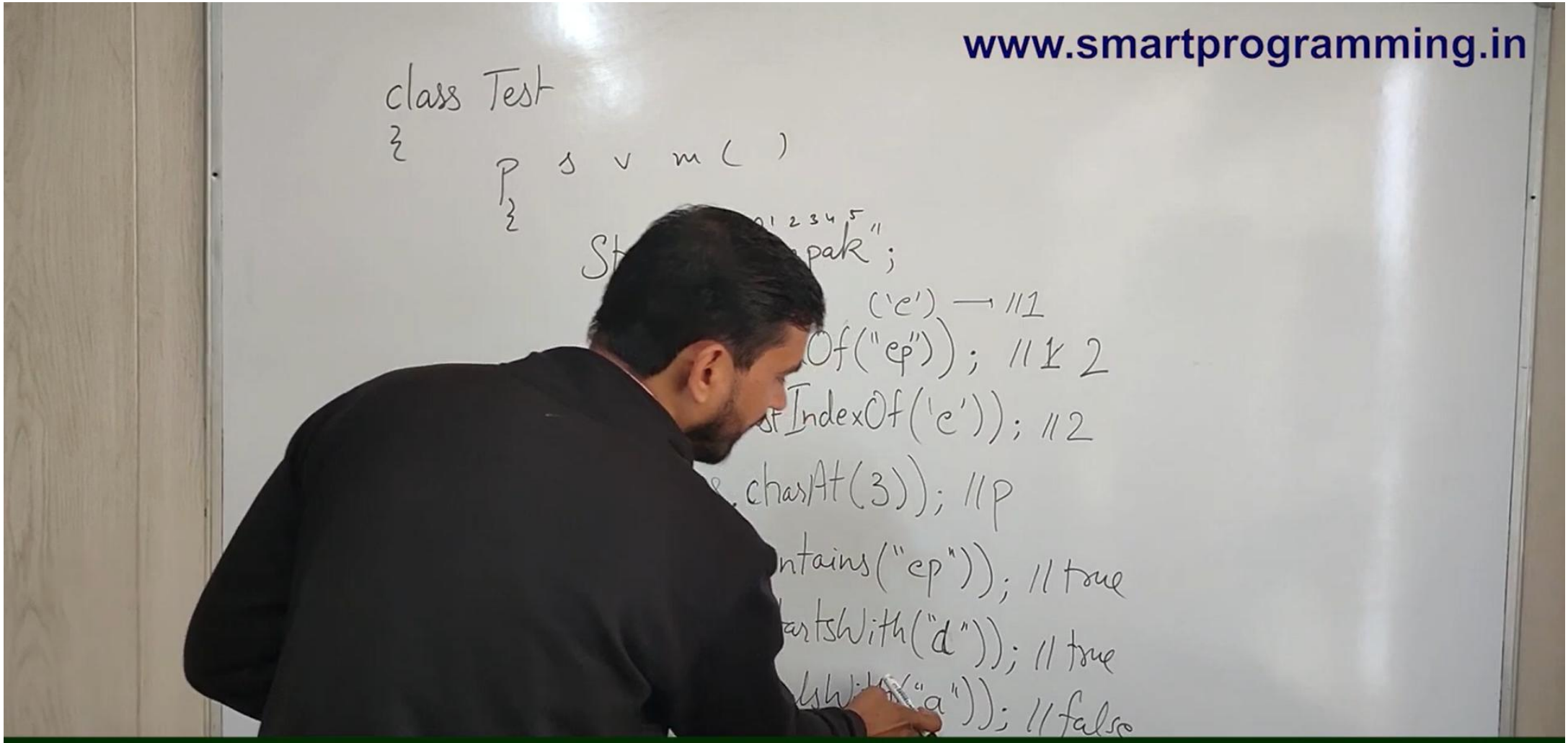
The charAt() method returns the character at the specified index. The index value should lie between 0 and length()-1.



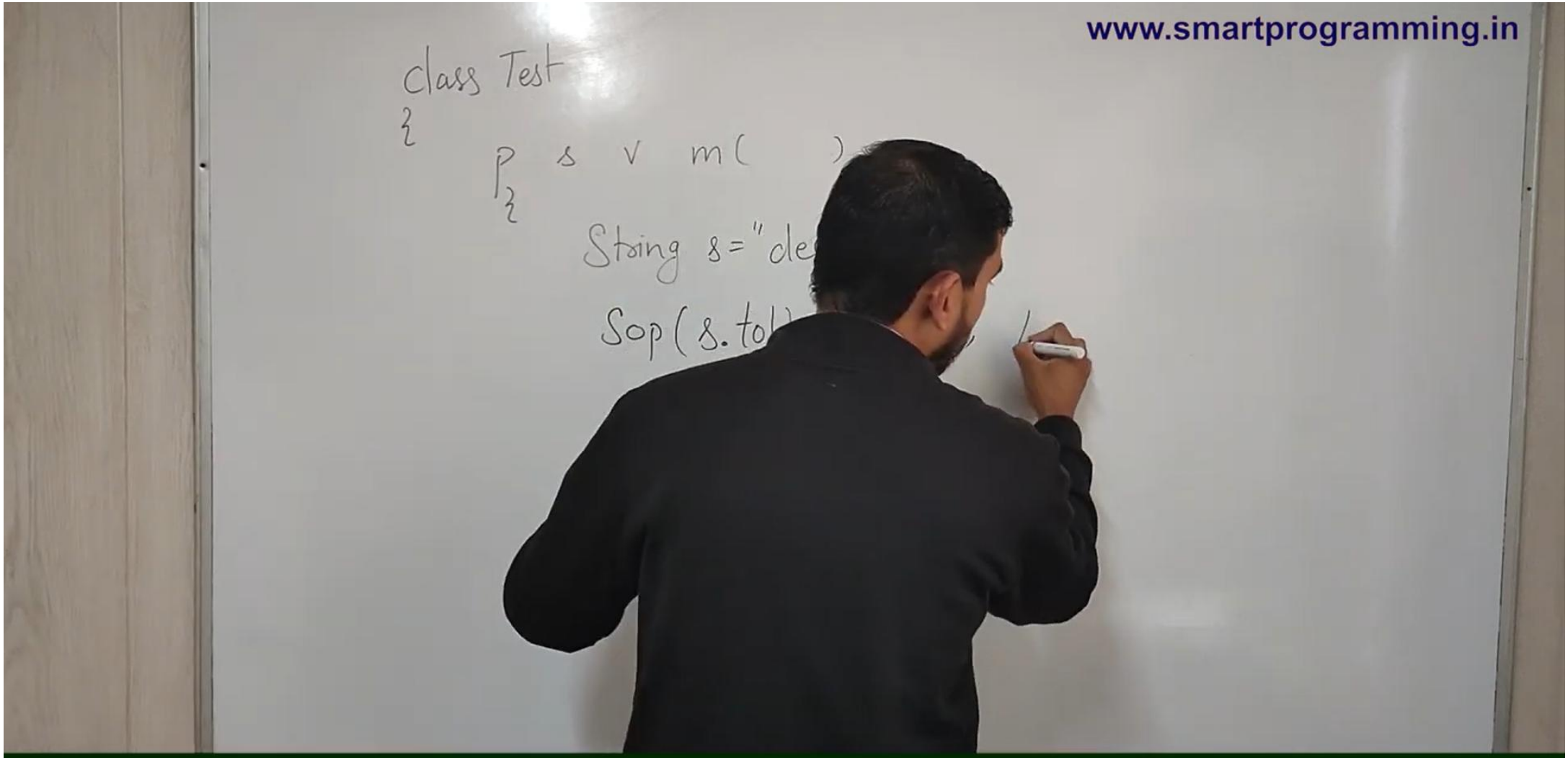
The contains() method searches the sequence of characters in the given string. It returns true if sequence of char values are found in this string otherwise returnsfalse.



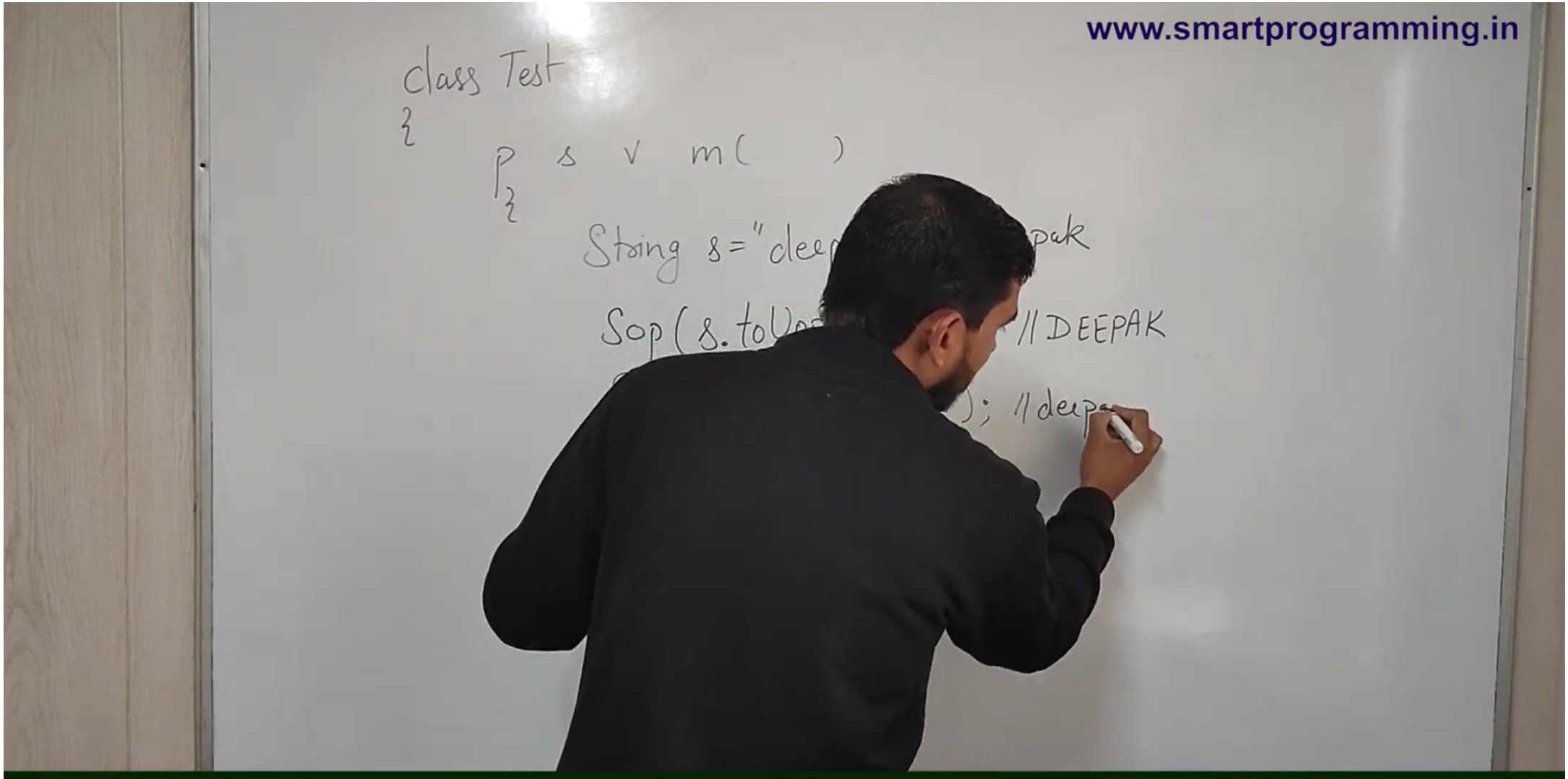
The startsWith() method tests if a string starts with the specified prefix beginning from 1st index. If yes then it will return true else it will return false.



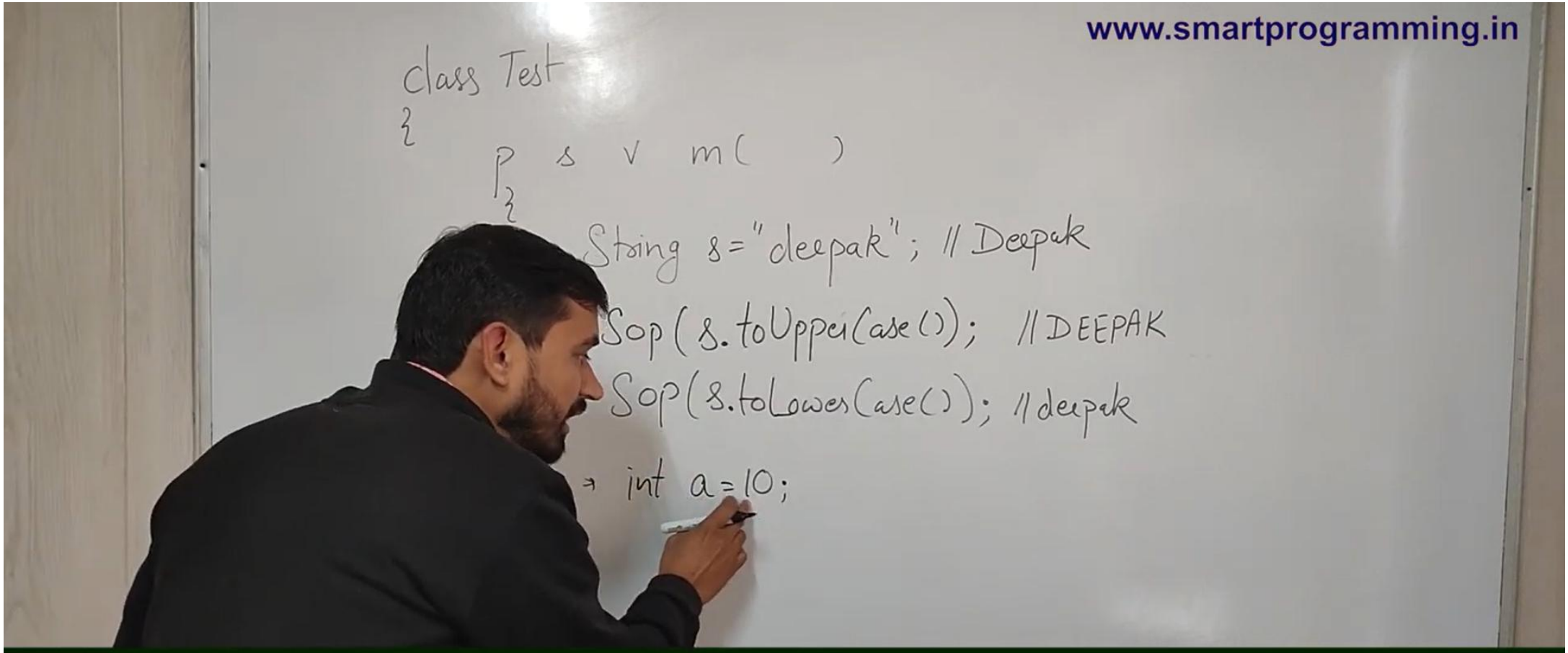
The endsWith() method checks whether the string ends with a specified suffix.
If yes, then method will return true otherwise it will return false.



The toUpperCase() method converts all characters of the string into a uppercase letter.



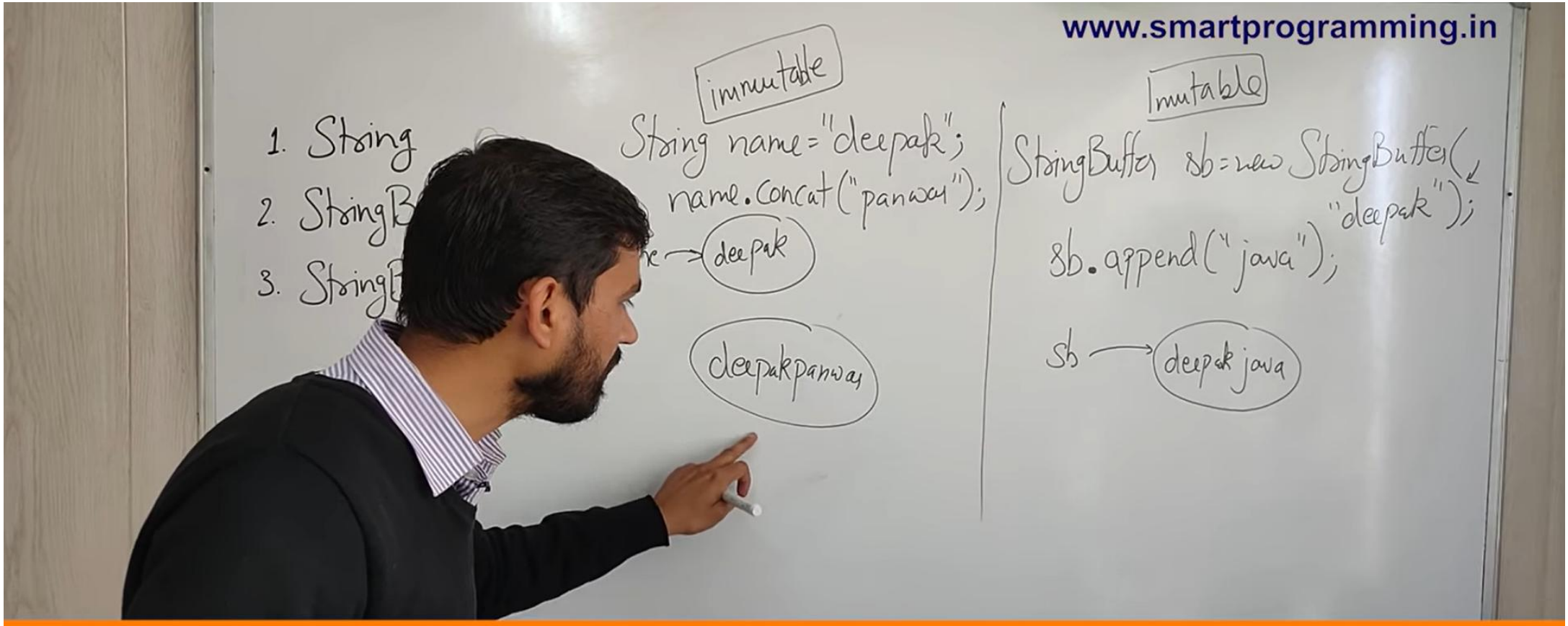
The toLowerCase() method converts all characters of the string into lowercase letter.



The valueOf() method converts different types of values into string. By the help of String.valueOf() method, we can convert int or long or float or double or object or any other type into string.
NOTE : valueOf() method is static method thats why we can call valueOf() method directly by String class.

Main difference between String and StringBuffer is :

String Objects are immutable and StringBuffer Objects are mutable



When we should use String and StringBuffer ?

If the data does not change or change one or two times only, use String.

If data is constantly or friquently changing like in calculator, notepad etc, we should use StringBuffer

```
public final class StringBuffer extends AbstractStringBuilder implements java.io.Serializable, CharSequence
{
    //4 constructors
    Stringbuffer() { ---- }
    StringBuffer(CharSequence seq) { ---- }
    StringBuffer(String str) { ---- }
    StringBuffer(int capacity) { ---- }

    //methods
    public synchronized int length( ) { ---- }
    public synchronized int capacity( ) { ---- }
    public synchronized StringBuffer append( ) { ---- }
    public synchronized StringBuffer insert( ) { ---- }
    public synchronized StringBuffer reverse( ) { ---- }
    public synchronized StringBuffer delete( ) { ---- }
    public synchronized StringBuffer deleteCharAt( ) { ---- }
    public synchronized StringBuffer replace( ) { ---- }
    public synchronized void ensureCapacity( ) { ---- }
    public synchronized char charAt() { ---- }
    public synchronized int indexOf() { ---- }
    public synchronized int lastIndexOf() { ---- }
    public synchronized String substring() { ---- }
    public synchronized CharSequence subSequence() { ---- }
    public synchronized String toString() { ---- }
}
```

StringBuffer class does not override equals method of Object class but String class override the equals method of Object class

StringBufferDemo - NetBeans IDE 8.0.2

www.smartprogramming.in

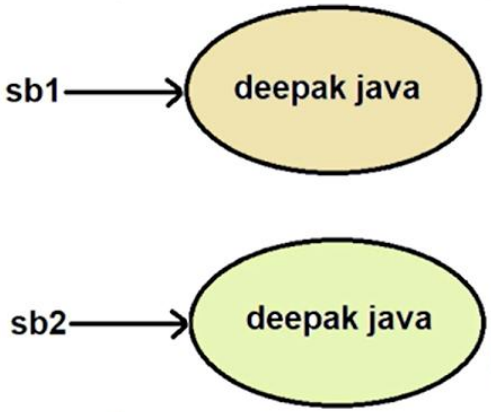
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

StringBufferDemo.java

Source History

```
1
2   package stringbufferdemo;
3
4   public class StringBufferDemo
5   {
6       public static void main(String[] args)
7       {
8           StringBuffer sb1=new StringBuffer("deepak java");
9           StringBuffer sb2=new StringBuffer("deepak java");
10          System.out.println(sb1.equals(sb2));
11      }
12  }
13
```



HTTP Server Monitor Output

9:25/1:

INS

76%

ENG 08:23 AM 20-01-2020

StringBufferDemo - NetBeans IDE 8.0.2

www.smartprogramming.in

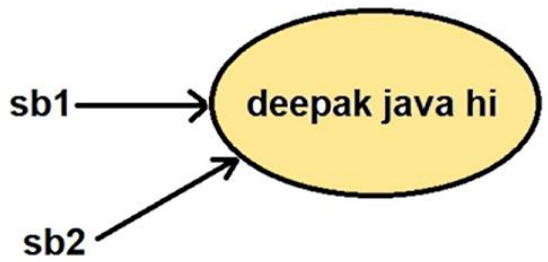
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

StringBufferDemo.java

Source History

```
1
2   package stringbufferdemo;
3
4   public class StringBufferDemo
5   {
6       public static void main(String[] args)
7       {
8           StringBuffer sb1=new StringBuffer("deepak java");
9           StringBuffer sb2=sb1.append("hi");
10          System.out.println(sb1.equals(sb2));
11      }
12  }
13
```



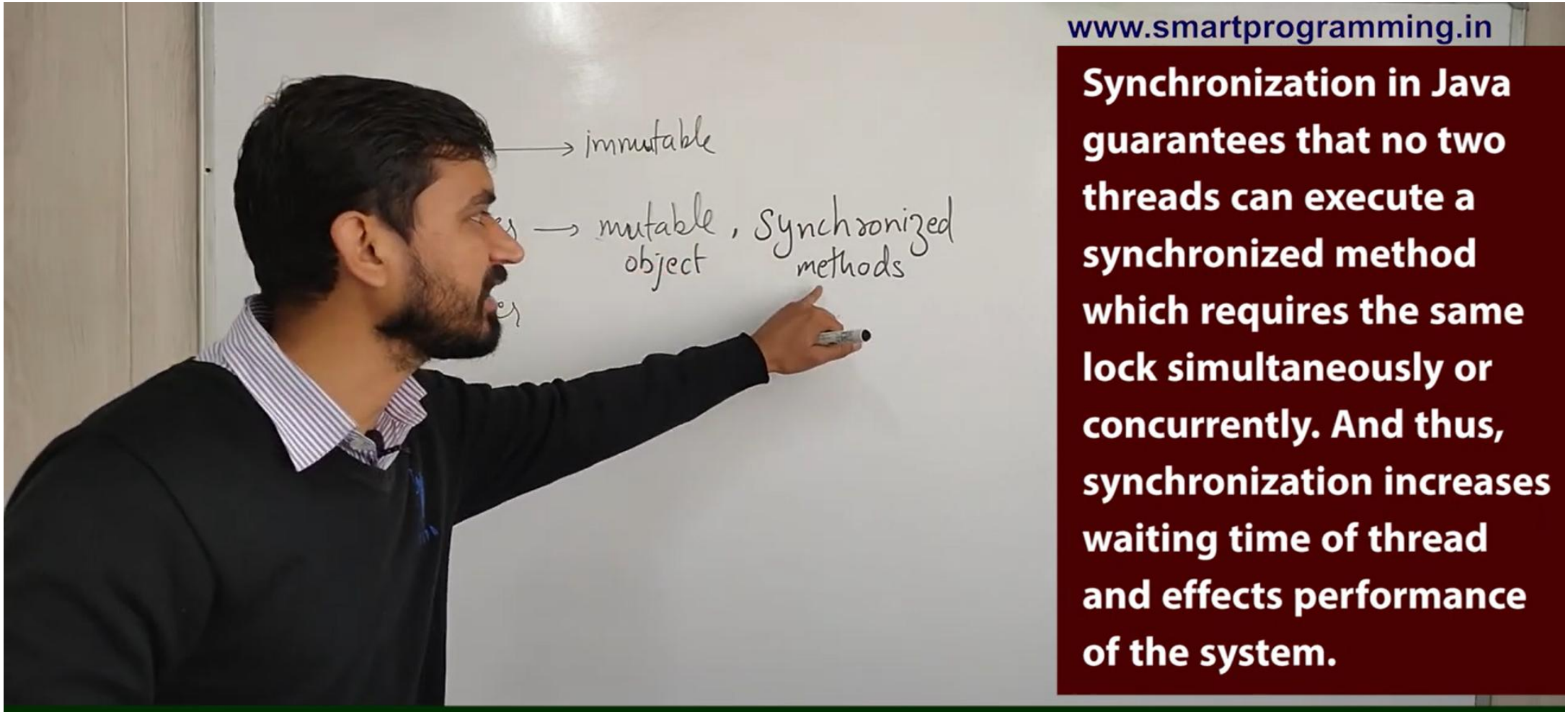
HTTP Server Monitor Output

9:25/1:

INS

75%

ENG 08:23 AM 20-01-2020

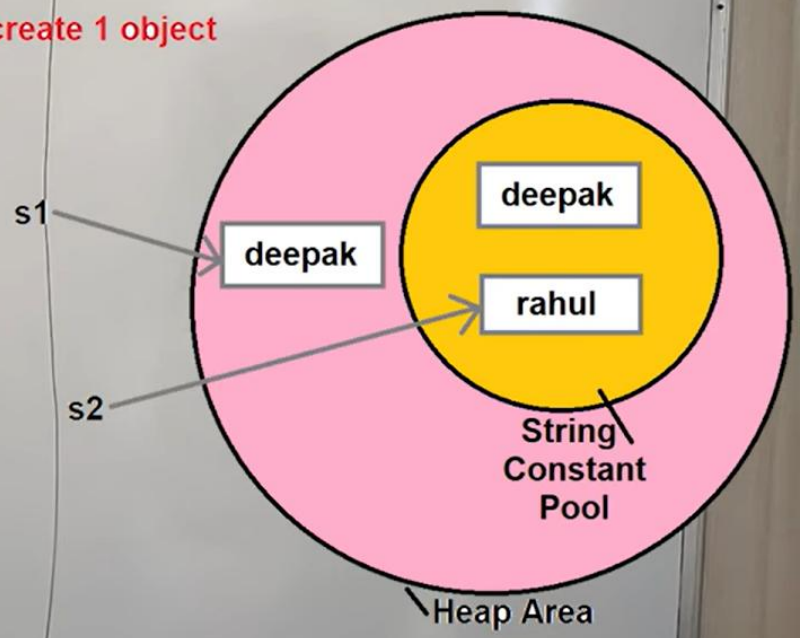

www.smartprogramming.in
immutable
mutable object , Synchronized methods
Synchronization in Java guarantees that no two threads can execute a synchronized method which requires the same lock simultaneously or concurrently. And thus, synchronization increases waiting time of thread and effects performance of the system.
To overcome the problem of slow performance of StringBuffer methods, Java introduced StringBuilder concept in JDK 1.5 version and creates all the methods of StringBuilder as non-synchronized which increases the methods performance.

```
public final class StringBuilder extends AbstractStringBuilder implements java.io.Serializable, CharSequence
{
    //4 constructors
    StringBuilder() { ---- }
    StringBuilder(CharSequence seq) { ---- }
    StringBuilder(String str) { ---- }
    StringBuilder(int capacity) { ---- }

    //methods
    public int length( ) { ---- }
    public int capacity( ) { ---- }
    public StringBuilder append( ) { ---- }
    public StringBuilder insert( ) { ---- }
    public StringBuilder reverse( ) { ---- }
    public StringBuilder delete( ) { ---- }
    public StringBuilder deleteCharAt( ) { ---- }
    public StringBuilder replace( ) { ---- }
    public void ensureCapacity( ) { ---- }
    public char charAt() { ---- }
    public int indexOf() { ---- }
    public int lastIndexOf() { ---- }
    public String substring() { ---- }
    public CharSequence subSequence() { ---- }
    public String toString() { ---- }
}
```

String	StringBuffer	StringBuilder
Storage	Heap area, SCP	

```
String s1=new String("deepak");  //it will create 2 objects
String s2="rahul";    //it will create 1 object
```



	String	StringBuffer	StringBuilder
Storage	Heap area, SCP	Heap area	Heap area
Objects	immutable object		

```
String s1=new String("deepak");   //it will create 2 objects
String s2="rahul";    //it will create 1 object
```