Oop's(object oriented programming system/structure)

1. **Oop is a programming paradigm / methodology**
    A. Oop
    B. Procedural paradigm
    C. Functional
    D. Logical
    E. Structural

2. 6 main pillars of oop are

1. Class
2. Inheritance
3. Abstraction
4. Object and methods
5. Polymorphism
6. encapsulation

3. Class

1. Class is the collection of objects.
2. Class is not a real world entity . it is just a template or blueprint or prototype.
3. Class does not occupy memory.
4. **Syntex :-**

```
 Access - modifier class ClassName{
        -method
        -constructors
        -fields
        -blocks
        -nested class
}
```

4. Method

1. A set of codes which perform a particular task.
2. **Advantages :**
    1. Code reusability
    2. Code optimization
3 **Syntax**

```
    access -modifier return type methodName(list of parameter){
    body
    }
```

5. Object

1. Object is a instance of class
2. Object is real world entity
3. Objects occupy memory.
4. Object consisted of identity like name
5. State / attribute like color, bread , age.
6. Behavior like eat , run.
7. **How to create an object**

1. By using new Keyboard
2. By using newInstance() method
3. By using clone() method.
4. By using deserialization.
5. By using factory() method.

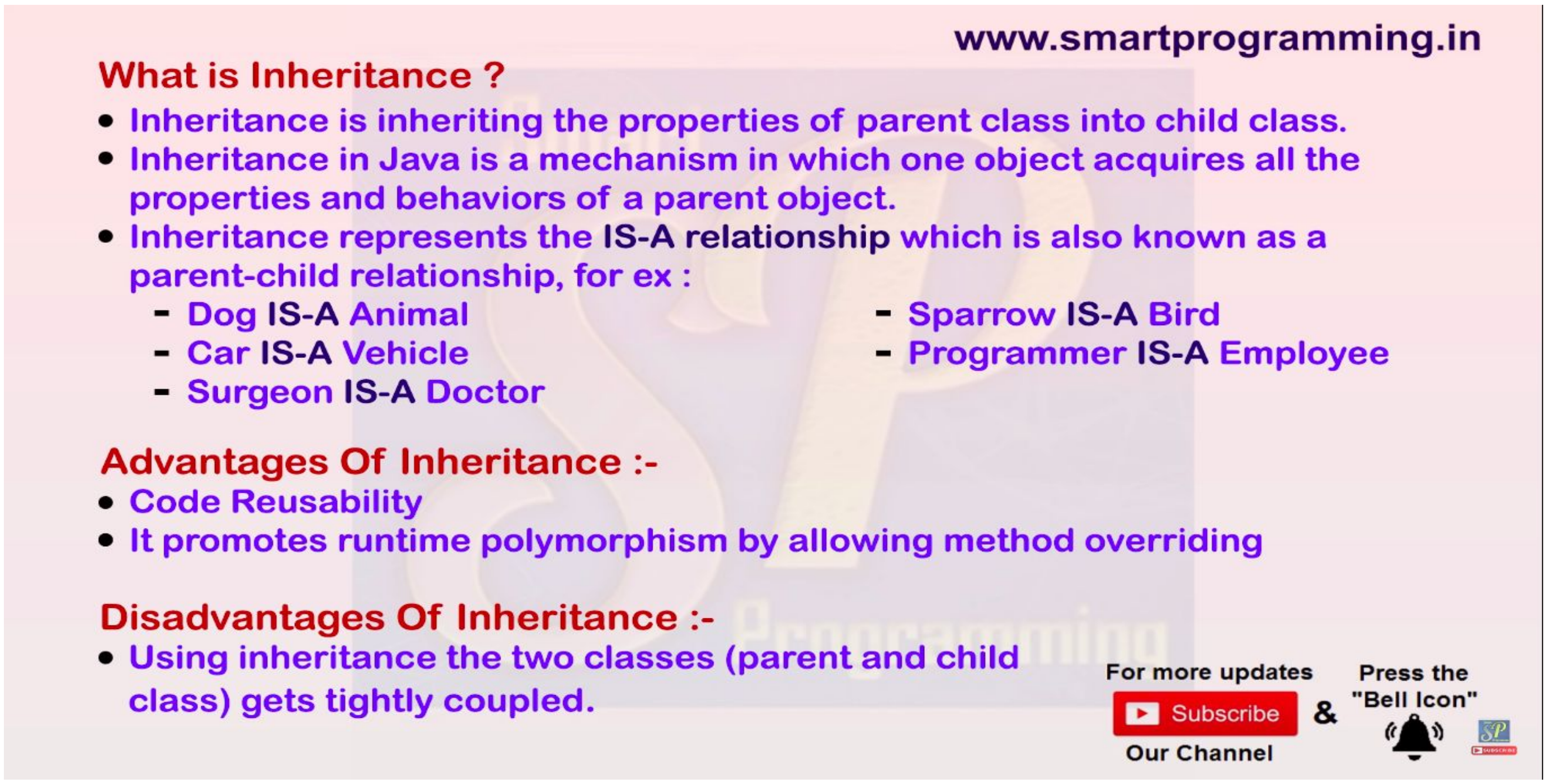
4. **Object initialization**
    1. By using reference variable
    2. Using method
    3. Using constructor

5. **Constructor**

    1. Is a block (similar to method) having the same name as that of class name.
    2. Does not have any return type not ever void.
    3. The only modifiers applicable for constructor are public , protected , default and private.
    4. It executes automatically when we create an object.

6. **Type of constructor**
    1. Default constructor (no argument constructor) created by compiler.
    2. No argument constructor created by the programmer.
    3. Parameterized constructor created by programmer.

7. Inheritance



www.smartprogramming.in

What is Inheritance ?

- Inheritance is inheriting the properties of parent class into child class.
- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship, for ex :
  - Dog IS-A Animal
  - Car IS-A Vehicle
  - Surgeon IS-A Doctor
  - Sparrow IS-A Bird
  - Programmer IS-A Employee

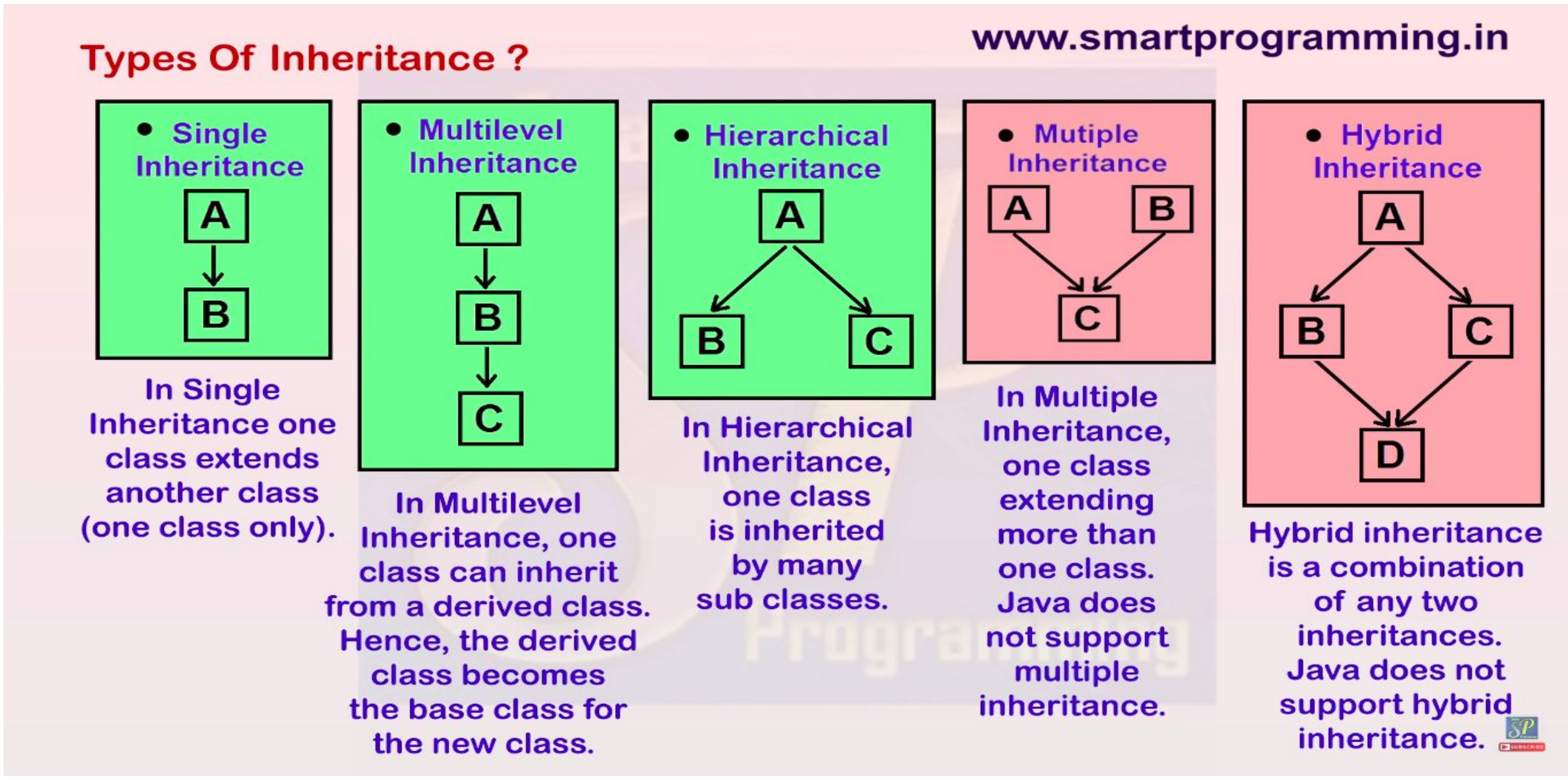
Advantages Of Inheritance :-

- Code Reusability
- It promotes runtime polymorphism by allowing method overriding

Disadvantages Of Inheritance :-

- Using inheritance the two classes (parent and child class) gets tightly coupled.

For more updates Subscribe Our Channel & Press the "Bell Icon"



www.smartprogramming.in

Types Of Inheritance ?

- Single Inheritance

$$\begin{tikzcd} A \arrow[d] \\ B \end{tikzcd}$$

In Single Inheritance one class extends another class (one class only).

- Multilevel Inheritance

$$\begin{tikzcd} A \arrow[d] \\ B \arrow[d] \\ C \end{tikzcd}$$

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.

- Hierarchical Inheritance

$$\begin{tikzcd} & A \arrow[dl] \arrow[dr] & \\ B & & C \end{tikzcd}$$

In Hierarchical Inheritance, one class is inherited by many sub classes.

- Mutiple Inheritance

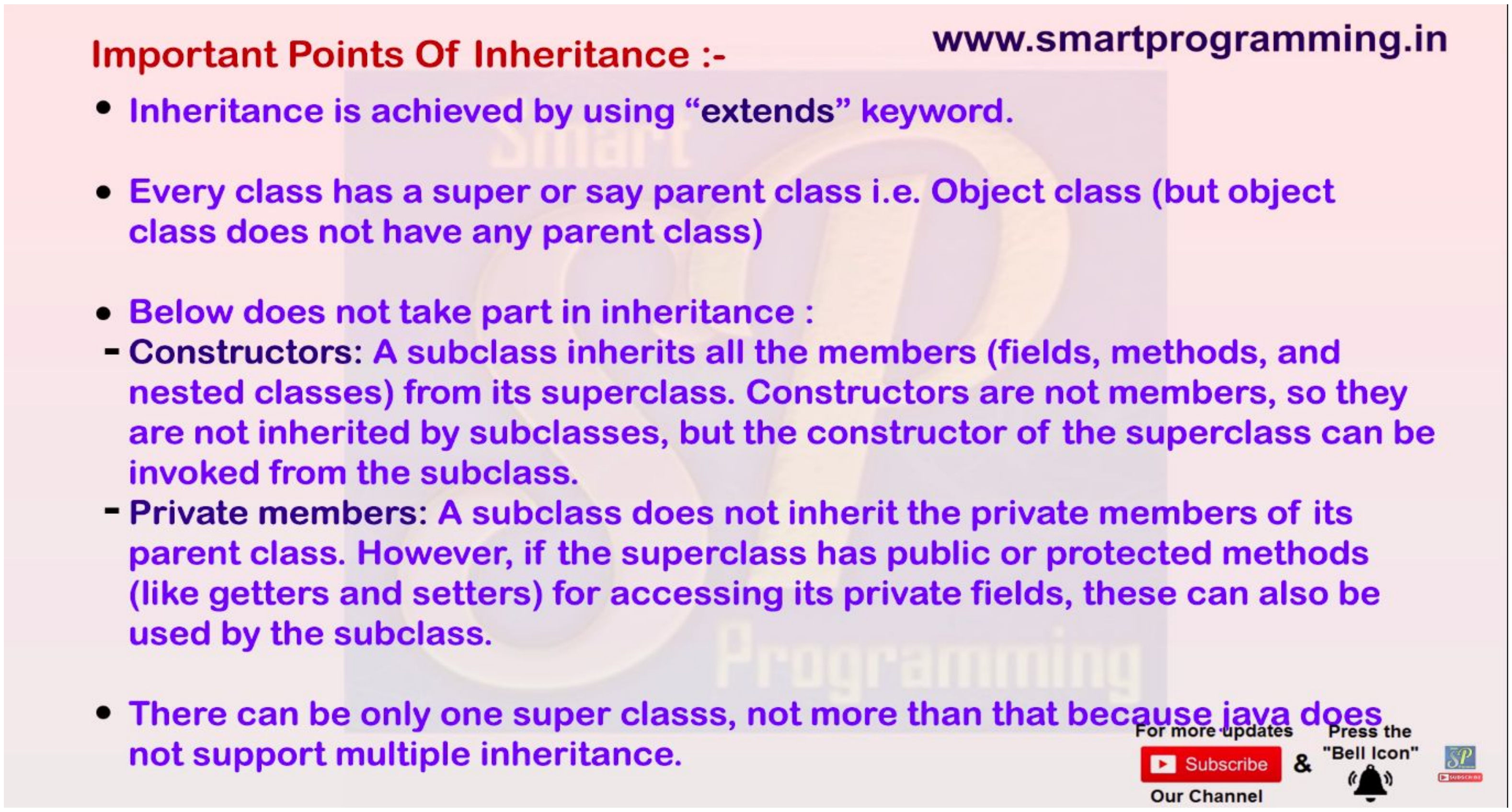
$$\begin{tikzcd} A \arrow[dr] & & B \arrow[dl] \\ & C & \end{tikzcd}$$

In Multiple Inheritance, one class extending more than one class. Java does not support multiple inheritance.

- Hybrid Inheritance

$$\begin{tikzcd} & A \arrow[dl] \arrow[dr] & \\ B \arrow[dr] & & C \arrow[dl] \\ & D & \end{tikzcd}$$

Hybrid inheritance is a combination of any two inheritances. Java does not support hybrid inheritance.



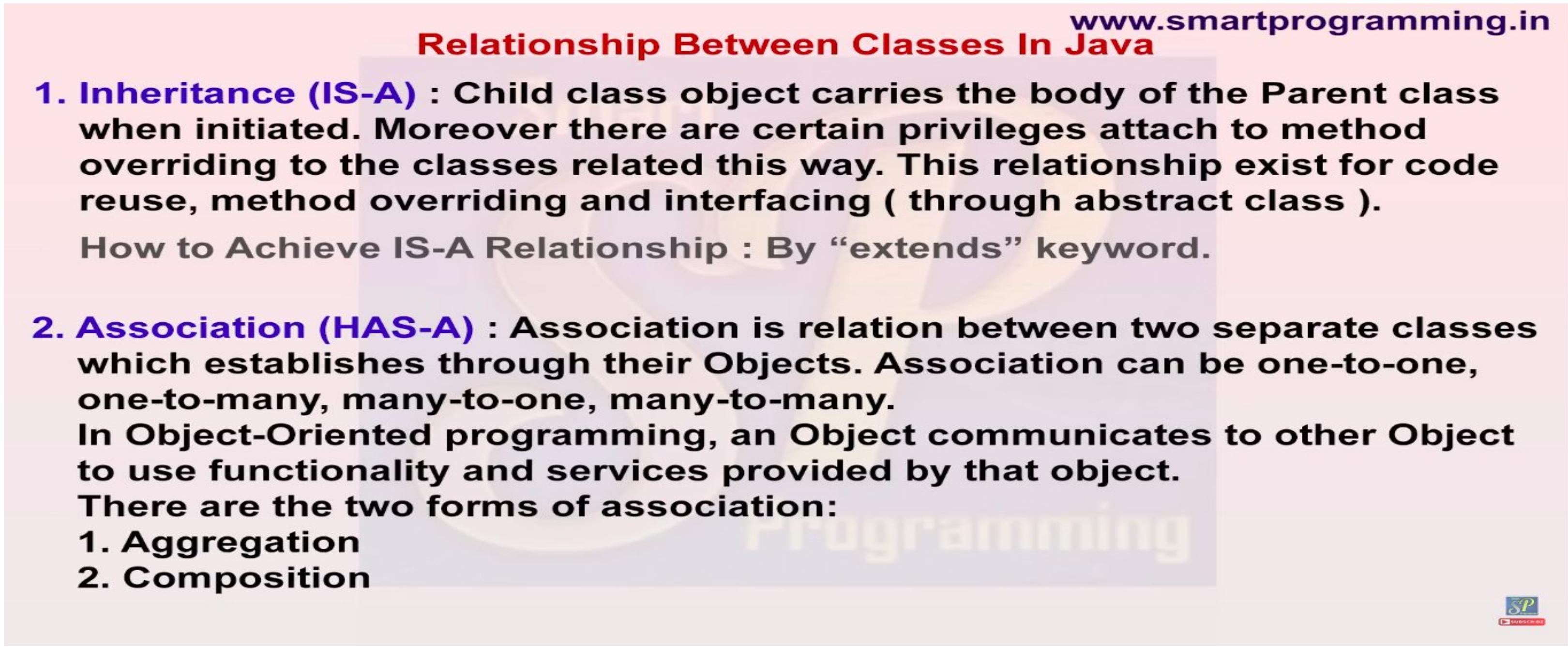
Important Points Of Inheritance :-

- Inheritance is achieved by using “extends” keyword.

- Every class has a super or say parent class i.e. Object class (but object class does not have any parent class)

- Below does not take part in inheritance :
  - Constructors: A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
  - Private members: A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods (like getters and setters) for accessing its private fields, these can also be used by the subclass.

- There can be only one super classs, not more than that because java does not support multiple inheritance.



Relationship Between Classes In Java

1. Inheritance (IS-A) : Child class object carries the body of the Parent class when initiated. Moreover there are certain privileges attach to method overriding to the classes related this way. This relationship exist for code reuse, method overriding and interfacing (through abstract class).

   How to Achieve IS-A Relationship : By “extends” keyword.

2. Association (HAS-A) : Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.
   In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object.
   There are the two forms of association:
   1. Aggregation
   2. Composition

Relationship Between Classes In Java

2.1. Aggregation : Without existing container object if there is a chance of existing contained objects , then container & contained objects are weakly associated & this weak association is known as aggregation.

For example : College consists of several professors, without existing college, there may be a chance of existing professor objects, hence college & professor objects are weakly associated & this is known as aggregation.

How to Achieve Aggregation : In aggregation container object holds just reference of contained objects.

For example :

```
final class College
{
        private Professor professor;
        void setProfessor(Professor professor)
        {
                this.professor = professor;
        }
        void teach()
        {
                if (professor != null)
                        professor.teach();
        }
}
```

With aggregation, the College also performs its functions through Professor, but the Professor is not always an internal part of the College. Professor may be swapped, or even completely removed. Not only that, but the outside world can still have a reference to the Professor, and tinker with it regardless of whether it's in the College.

Relationship Between Classes In Java

2.2. Composition : Without existing container object if there is no chance of existing contained objects , then container & contained objects are strongly associated & this strong association is known as composition.

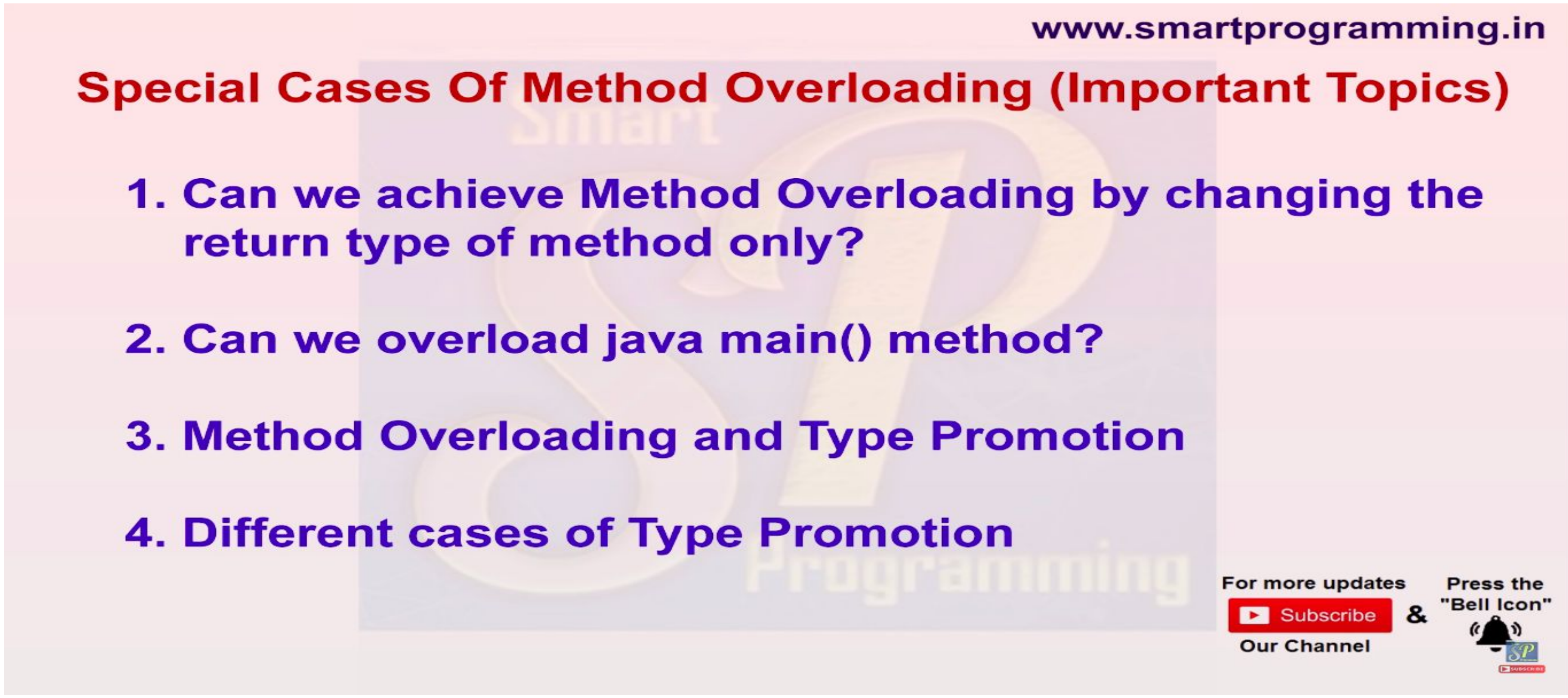
For example : College consists of several branches, without existing college, there is no chance of existing branches, hence college & branches are strongly associated & this is known as composition.

How to Achieve Composition : In composition container object holds directly contained objects

For example :

```
final class College
{
        private final Branches branches;
        College(BranchesNames names)
        {
                branches = new Branches(names);
        }
}
```

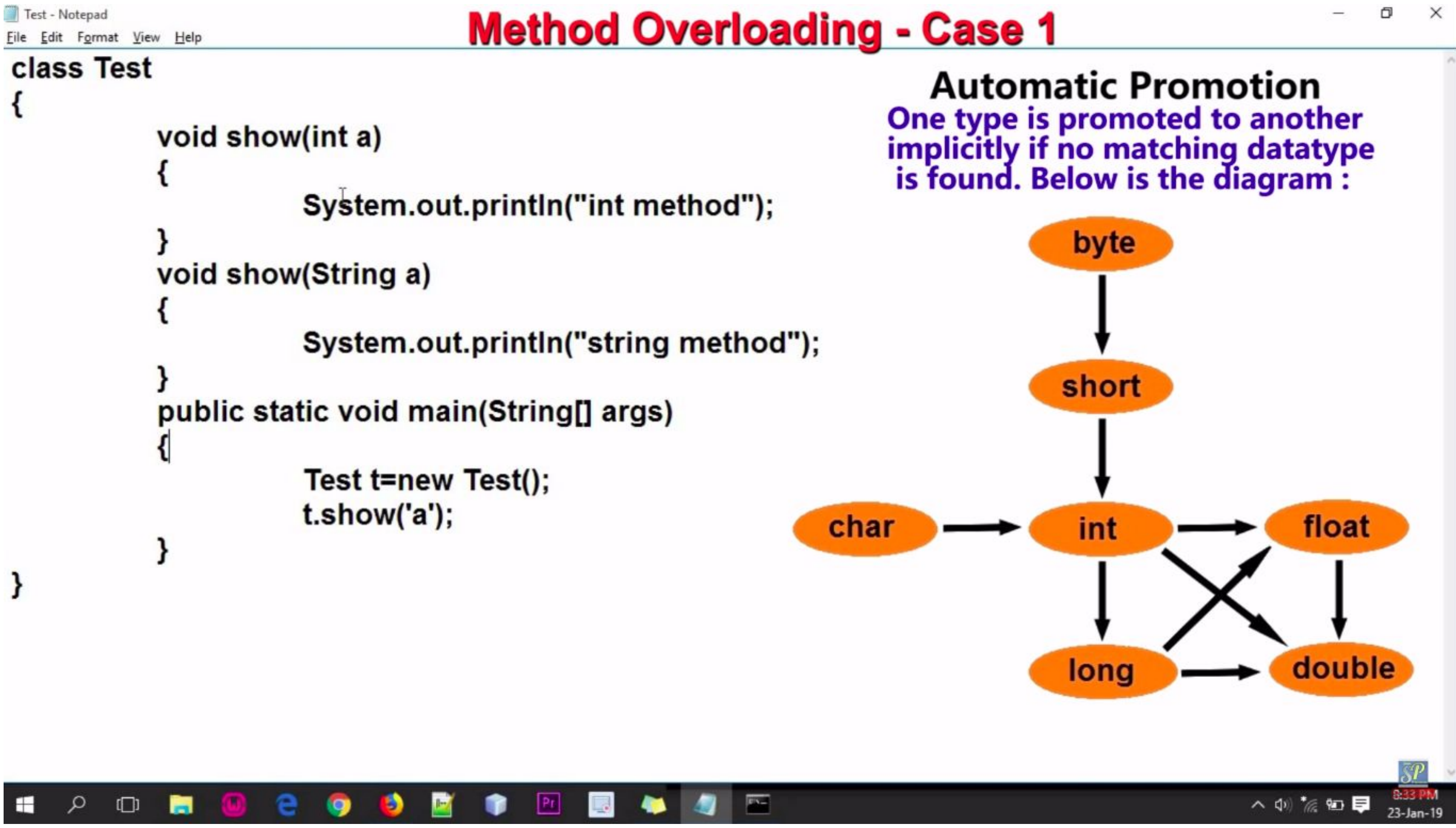
In the case of composition, the Branches is completely encapsulated by the College. There is no way for the outside world to get a reference to the Branches. The Branches lives and dies with the College.



www.smartprogramming.in

Special Cases Of Method Overloading (Important Topics)

1. Can we achieve Method Overloading by changing the return type of method only?
2. Can we overload java main() method?
3. Method Overloading and Type Promotion
4. Different cases of Type Promotion



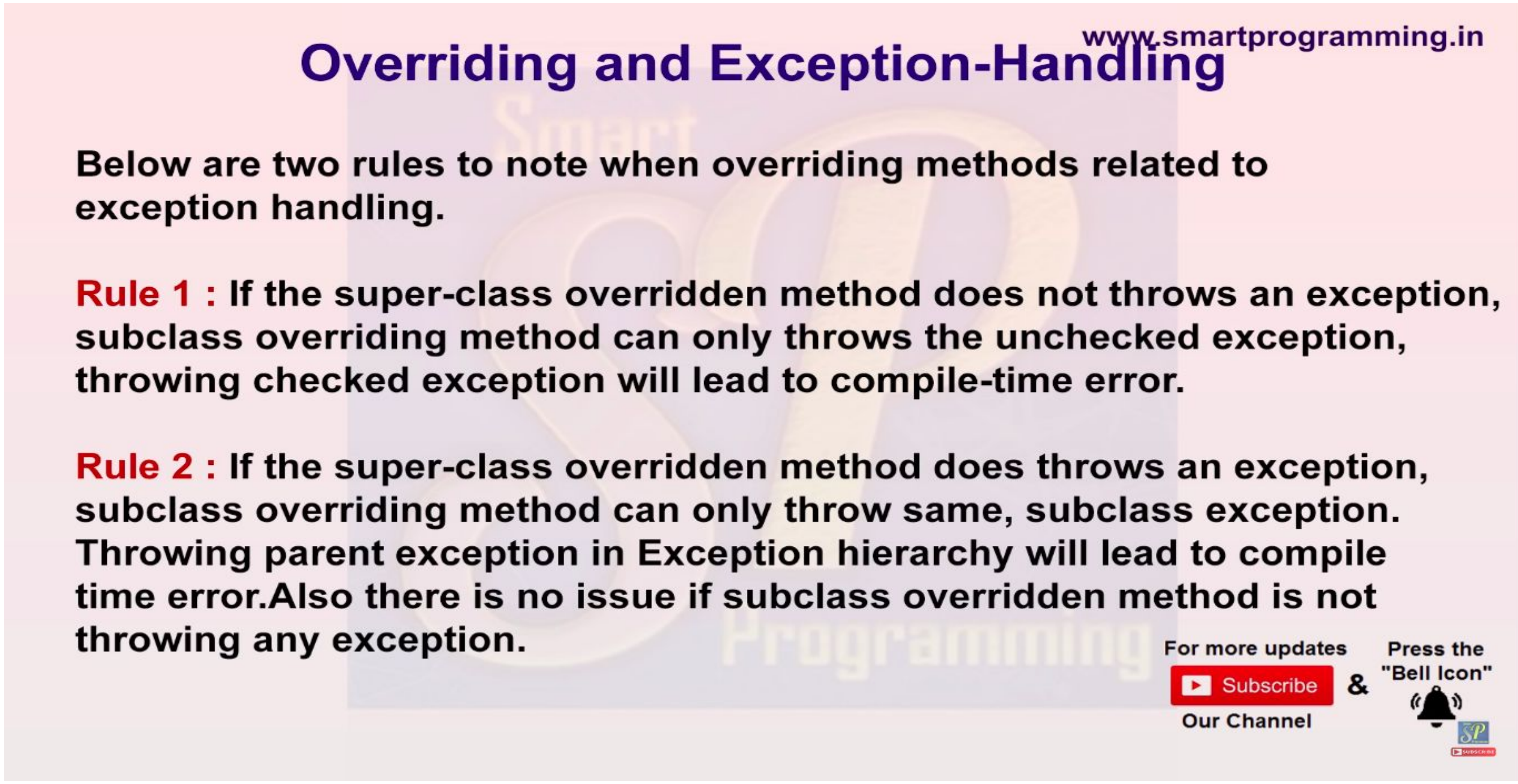
Method Overloading - Case 1

```
class Test
{
		void show(int a)
		{
				System.out.println("int method");
		}
		void show(String a)
		{
				System.out.println("string method");
		}
		public static void main(String[] args)
		{
				Test t=new Test();
				t.show('a');
		}
}
```

Automatic Promotion

One type is promoted to another implicitly if no matching datatype is found. Below is the diagram :

$$\begin{tikzcd}
 & \text{byte} \arrow[d] & \\
 & \text{short} \arrow[d] & \\
\text{char} \arrow[r] & \text{int} \arrow[r] \arrow[d] \arrow[dr] & \text{float} \arrow[d] \\
 & \text{long} \arrow[r] \arrow[ur] & \text{double}
\end{tikzcd}$$

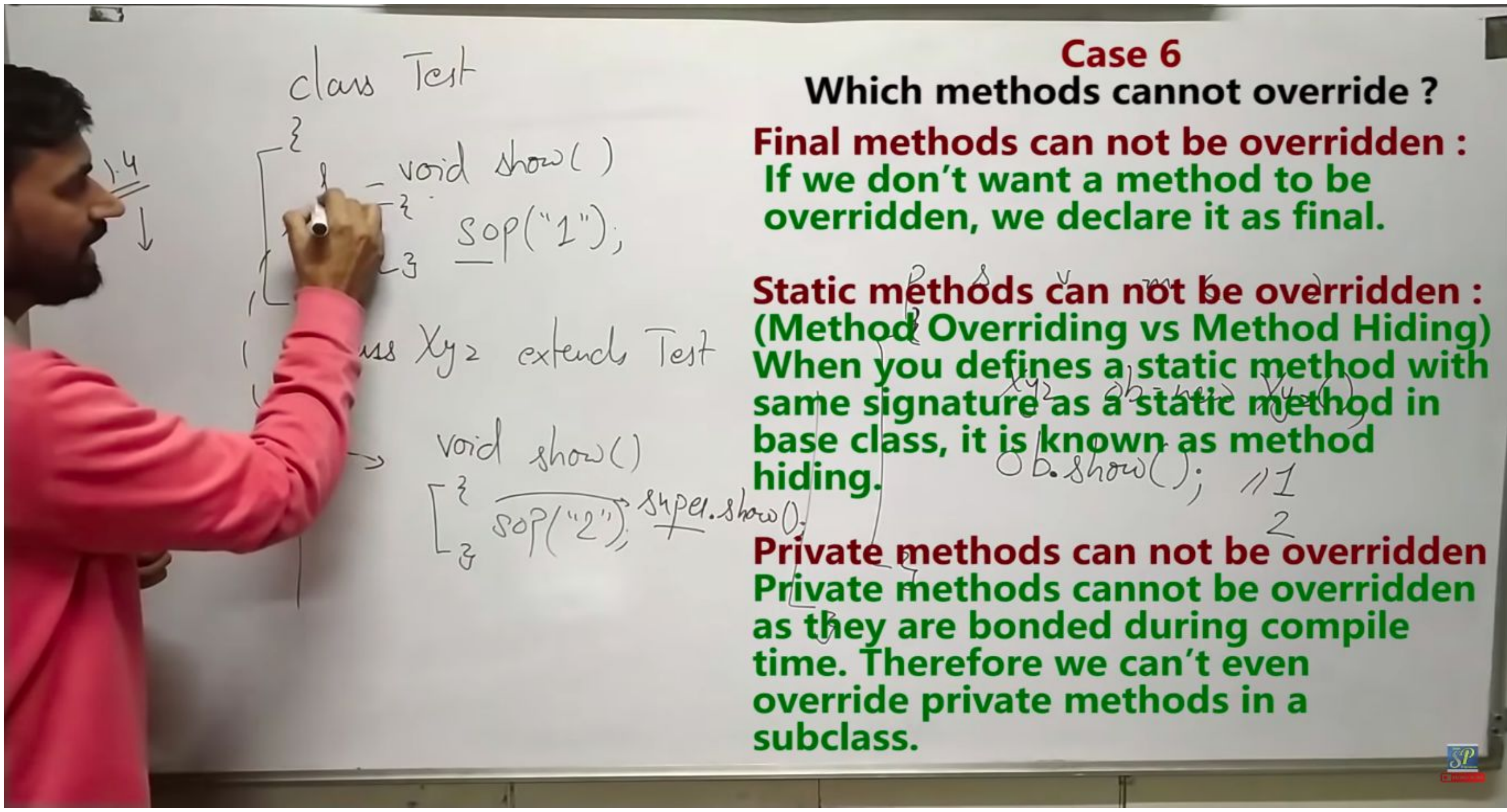


Overriding and Exception-Handling

Below are two rules to note when overriding methods related to exception handling.

Rule 1 : If the super-class overridden method does not throws an exception, subclass overriding method can only throws the unchecked exception, throwing checked exception will lead to compile-time error.

Rule 2 : If the super-class overridden method does throws an exception, subclass overriding method can only throw same, subclass exception. Throwing parent exception in Exception hierarchy will lead to compile time error.Also there is no issue if subclass overridden method is not throwing any exception.

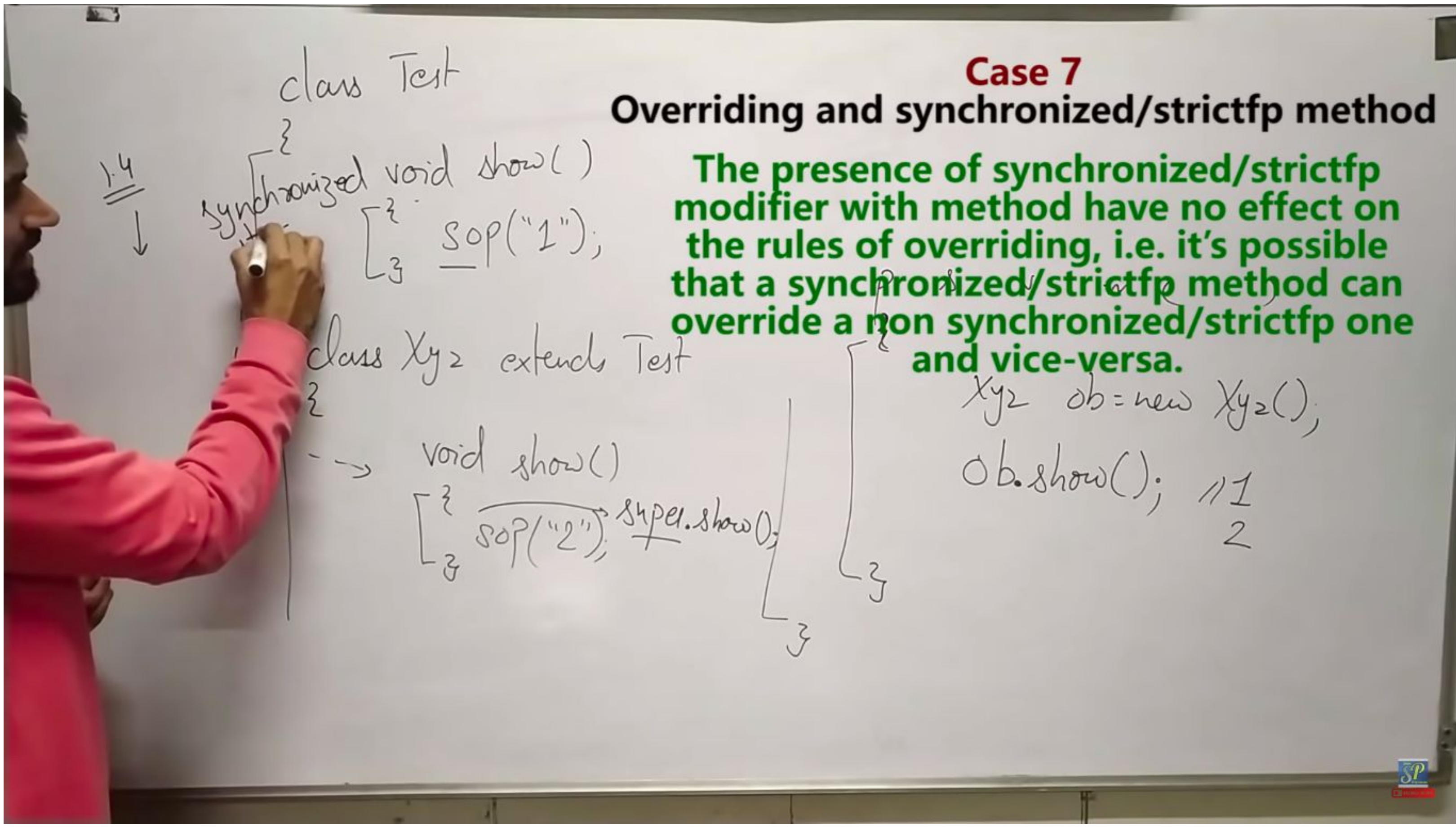


Case 6
Which methods cannot override ?

Final methods can not be overridden :
If we don't want a method to be overridden, we declare it as final.

Static methods can not be overridden :
(Method Overriding vs Method Hiding)
When you defines a static method with same signature as a static method in base class, it is known as method hiding.

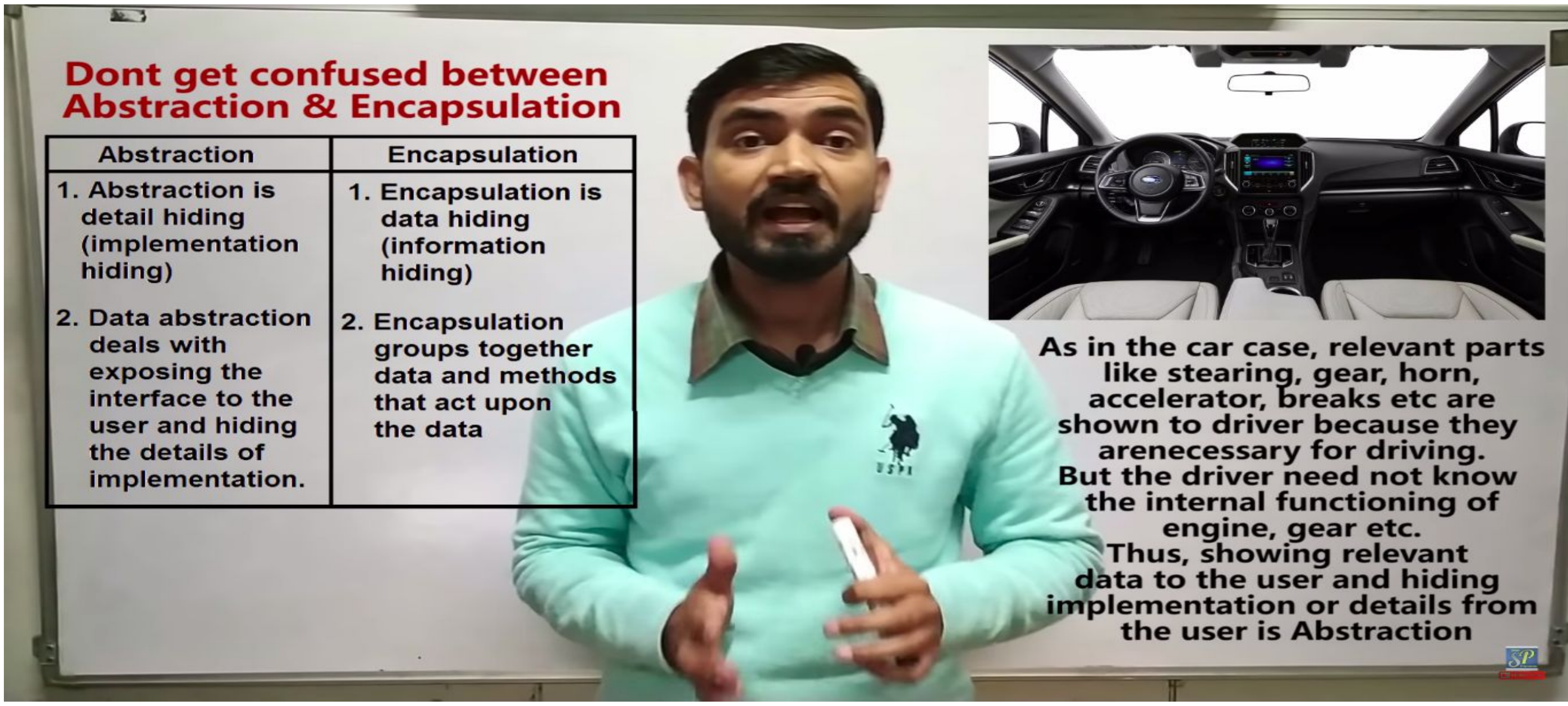
Private methods can not be overridden
Private methods cannot be overridden as they are bonded during compile time. Therefore we can't even override private methods in a subclass.



Case 7
Overriding and synchronized/strictfp method

The presence of synchronized/strictfp modifier with method have no effect on the rules of overriding, i.e. it's possible that a synchronized/strictfp method can override a non synchronized/strictfp one and vice-versa.

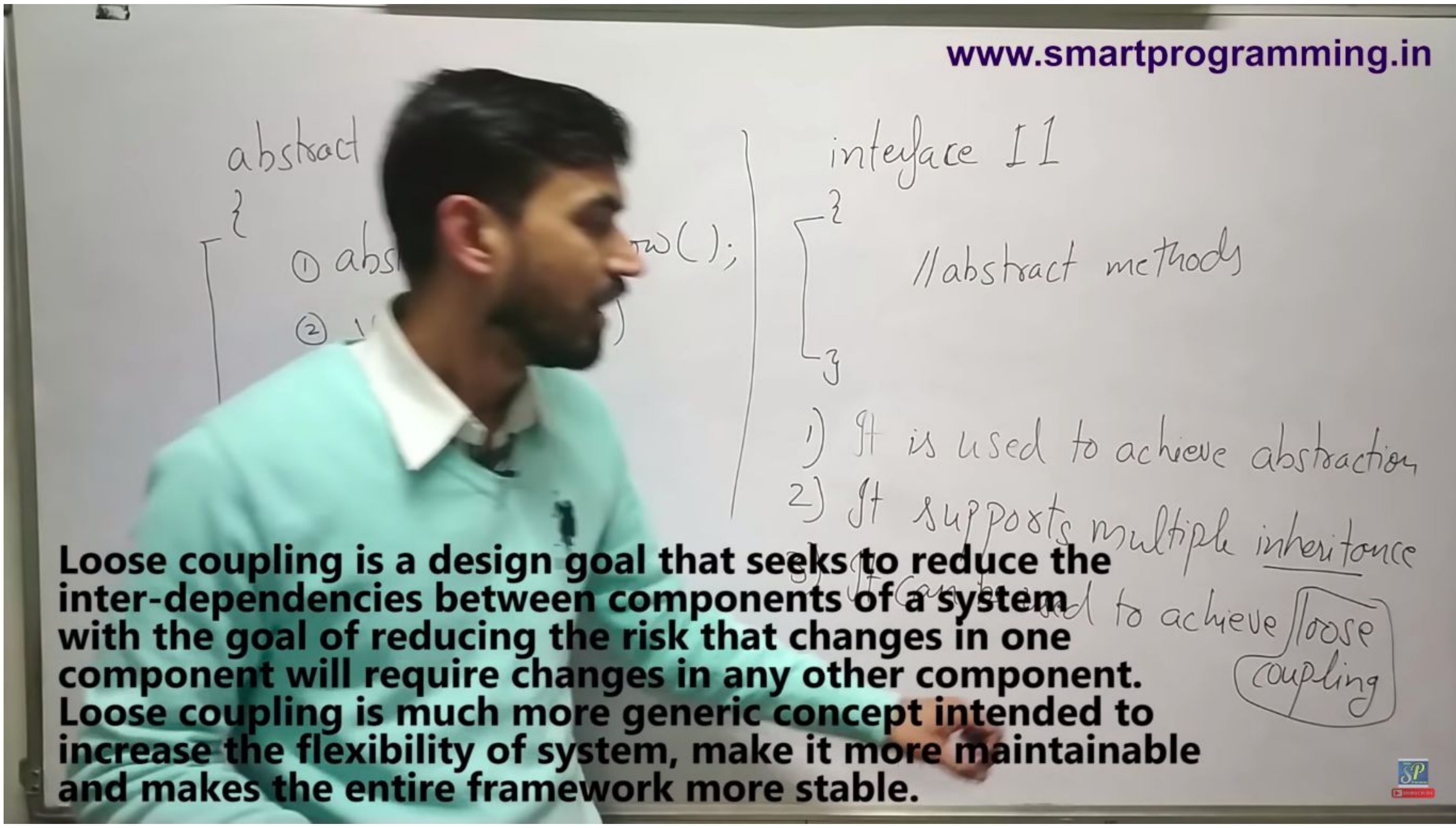
```
1.4
 ↓
class Test
{
synchronized void show()
 {
  sop("1");
 }
class Xyz extend Test
{
 --> void show()
  {
   sop("2"); super.show();
  }
}
{
 Xyz ob=new Xyz();
 ob.show(); //1
              2
}
```



Dont get confused between Abstraction & Encapsulation

Abstraction	Encapsulation
1. Abstraction is detail hiding (implementation hiding)	1. Encapsulation is data hiding (information hiding)
2. Data abstraction deals with exposing the interface to the user and hiding the details of implementation.	2. Encapsulation groups together data and methods that act upon the data

As in the car case, relevant parts like stearing, gear, horn, accelerator, breaks etc are shown to driver because they arenecessary for driving. But the driver need not know the internal functioning of engine, gear etc. Thus, showing relevant data to the user and hiding implementation or details from the user is Abstraction



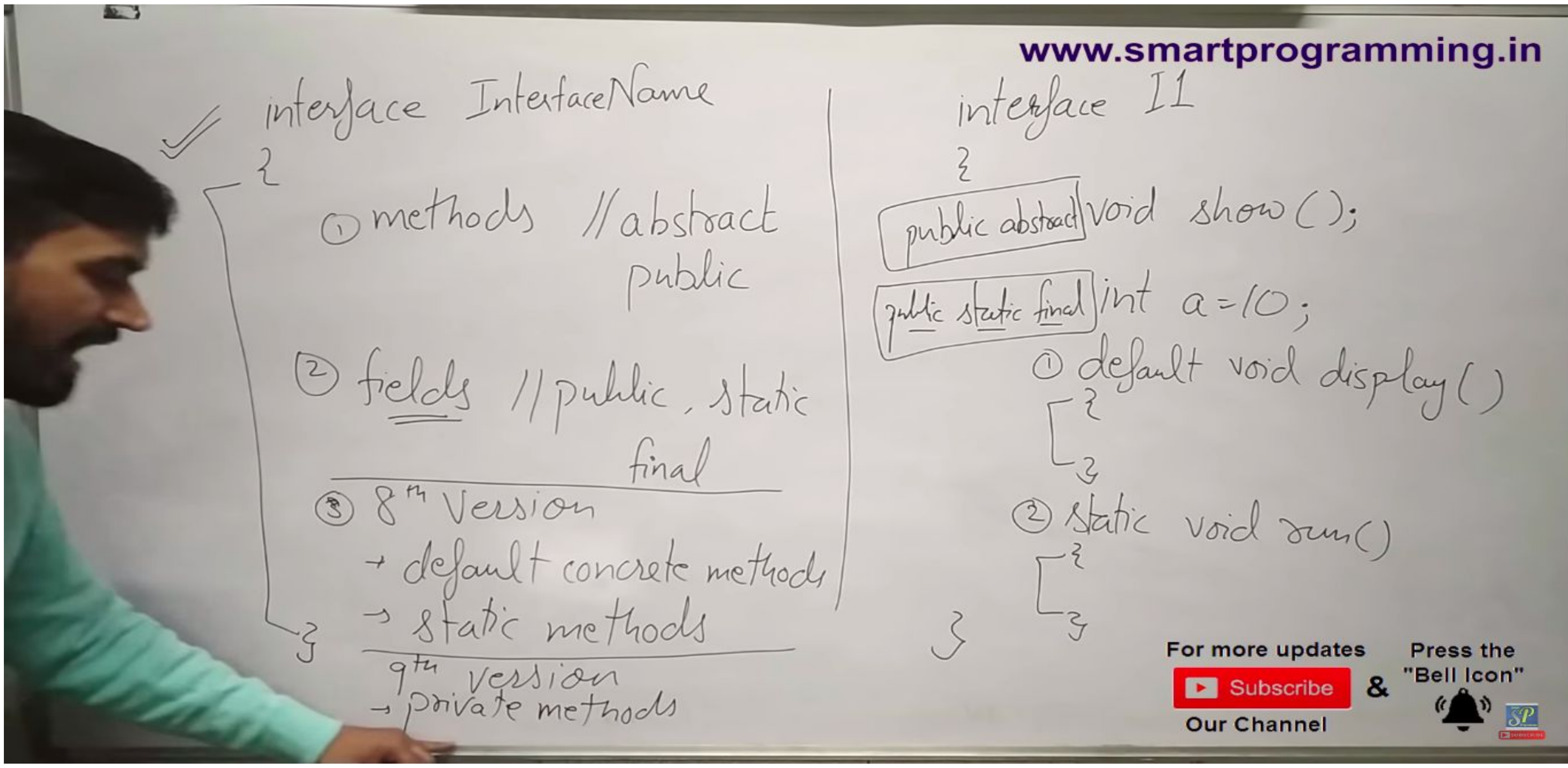
www.smartprogramming.in

```
abstract
{
  ① abs            ow();
  ②
```

```
interface I1
{
    //abstract methods
}
```

1) It is used to achieve abstraction
2) It supports multiple inheritance
3) It can be used to achieve loose coupling

Loose coupling is a design goal that seeks to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component. Loose coupling is much more generic concept intended to increase the flexibility of system, make it more maintainable and makes the entire framework more stable.

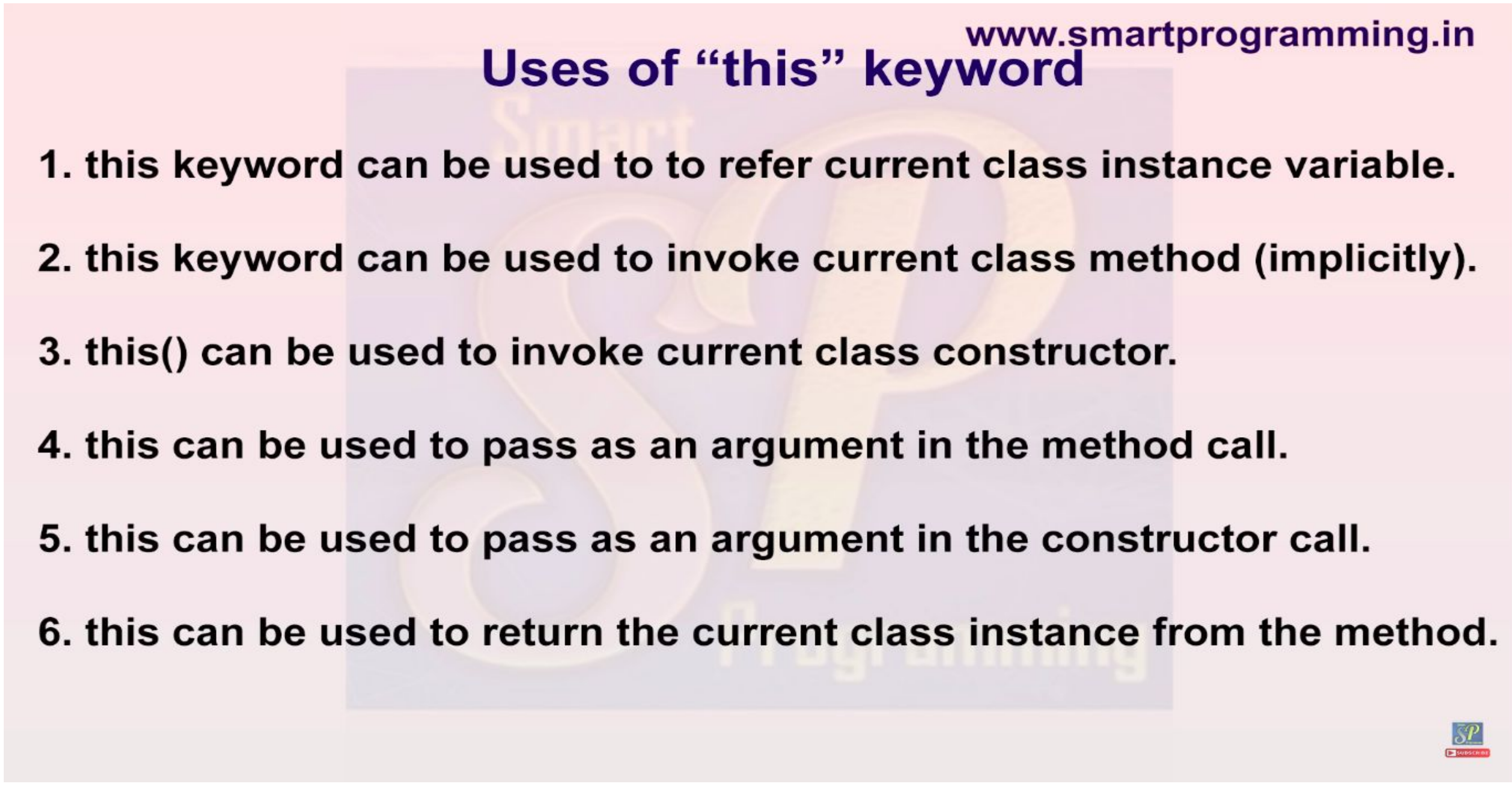


www.smartprogramming.in

```
interface InterfaceName
{
  ① methods //abstract
               public

  ② fields //public, static
               final
  ③ 8th Version
     → default concrete methods
     → static methods
}
     9th version
     → private methods
```

```
interface I1
{
  public abstract void show();
  public static final int a=10;
    ① default void display()
      {
      }
    ② static void sum()
      {
      }
}
```

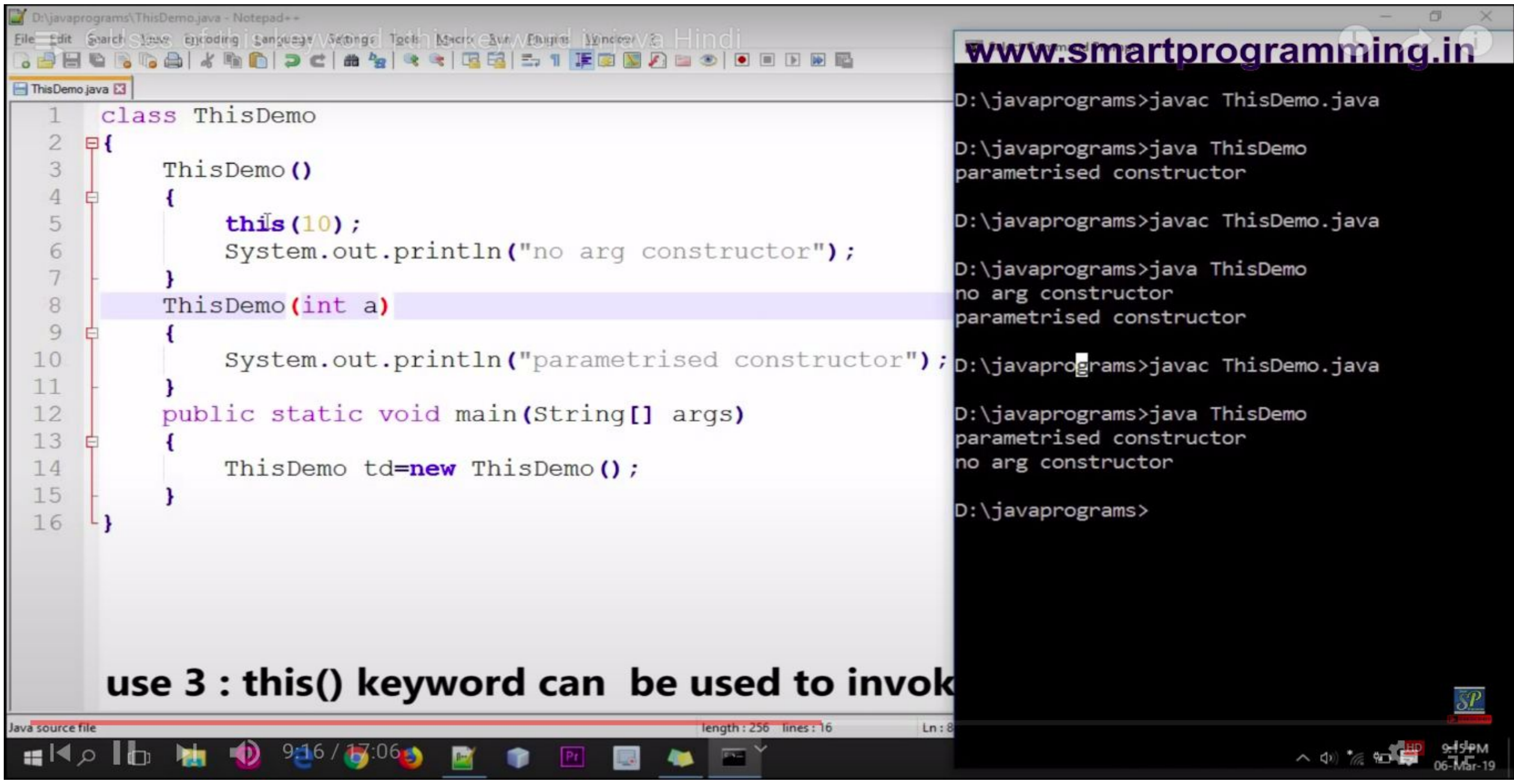
For more updates Subscribe Our Channel & Press the "Bell Icon"



www.smartprogramming.in

Uses of “this” keyword

1. this keyword can be used to to refer current class instance variable.
2. this keyword can be used to invoke current class method (implicitly).
3. this() can be used to invoke current class constructor.
4. this can be used to pass as an argument in the method call.
5. this can be used to pass as an argument in the constructor call.
6. this can be used to return the current class instance from the method.

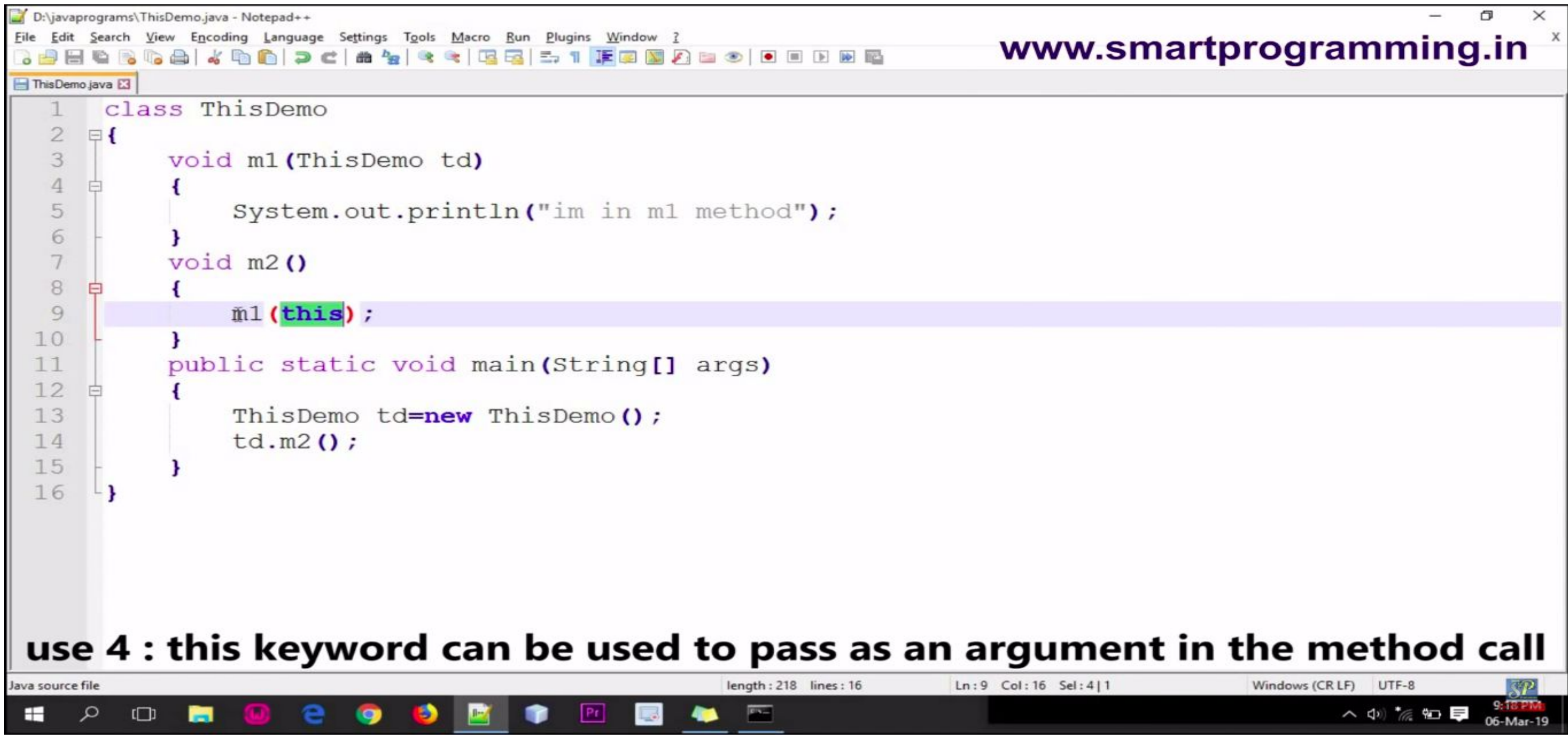


```java
class ThisDemo
{
	ThisDemo()
	{
		this(10);
		System.out.println("no arg constructor");
	}
	ThisDemo(int a)
	{
		System.out.println("parametrised constructor");
	}
	public static void main(String[] args)
	{
		ThisDemo td=new ThisDemo();
	}
}
```

www.smartprogramming.in

```
D:\javaprograms>javac ThisDemo.java

D:\javaprograms>java ThisDemo
parametrised constructor

D:\javaprograms>javac ThisDemo.java

D:\javaprograms>java ThisDemo
no arg constructor
parametrised constructor

D:\javaprograms>javac ThisDemo.java

D:\javaprograms>java ThisDemo
parametrised constructor
no arg constructor

D:\javaprograms>
```

use 3 : this() keyword can be used to invok

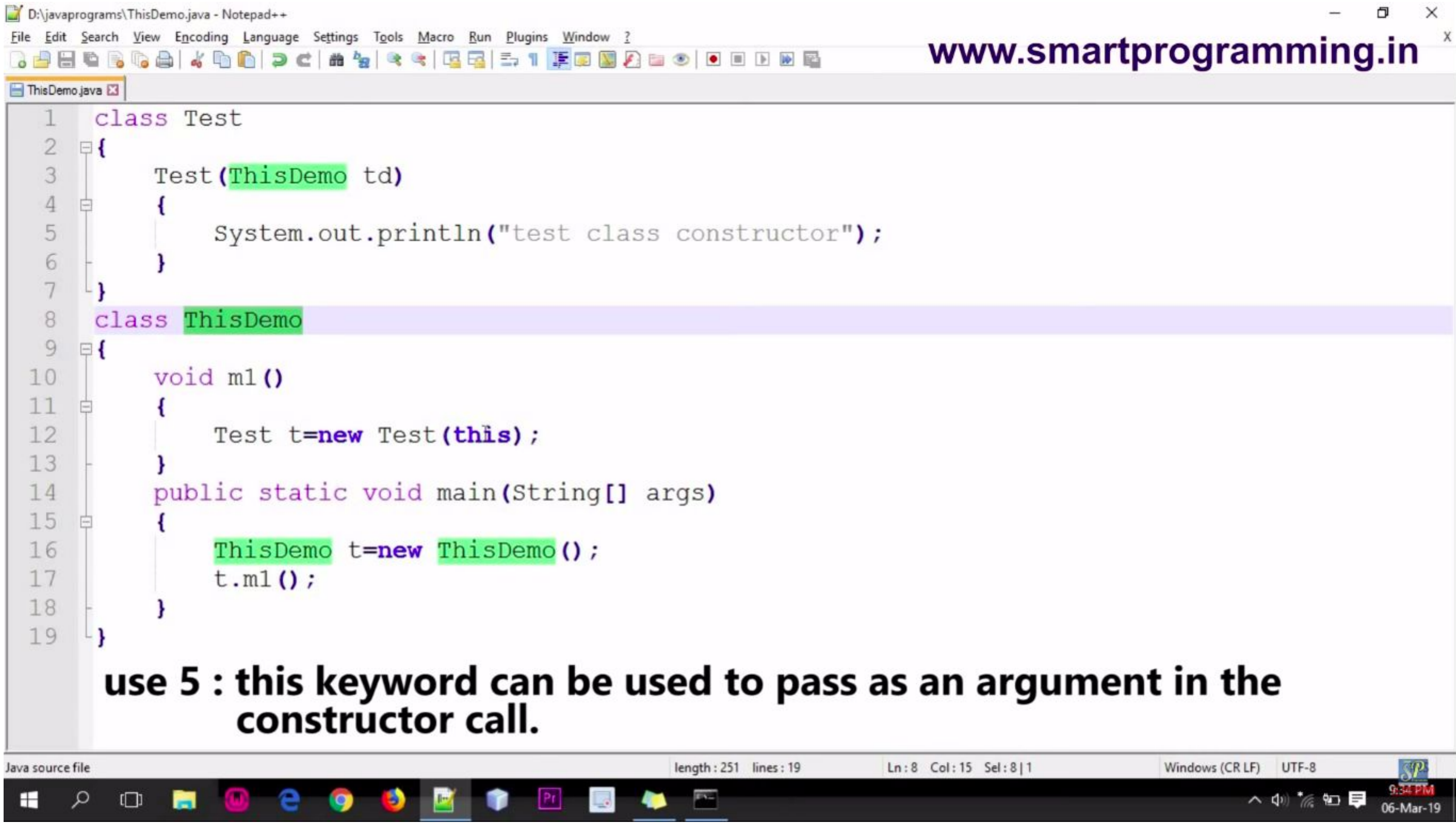


D:\javaprograms\ThisDemo.java - Notepad++

www.smartprogramming.in

```java
class ThisDemo
{
    void m1(ThisDemo td)
    {
        System.out.println("im in m1 method");
    }
    void m2()
    {
        m1(this);
    }
    public static void main(String[] args)
    {
        ThisDemo td=new ThisDemo();
        td.m2();
    }
}
```

use 4 : this keyword can be used to pass as an argument in the method call

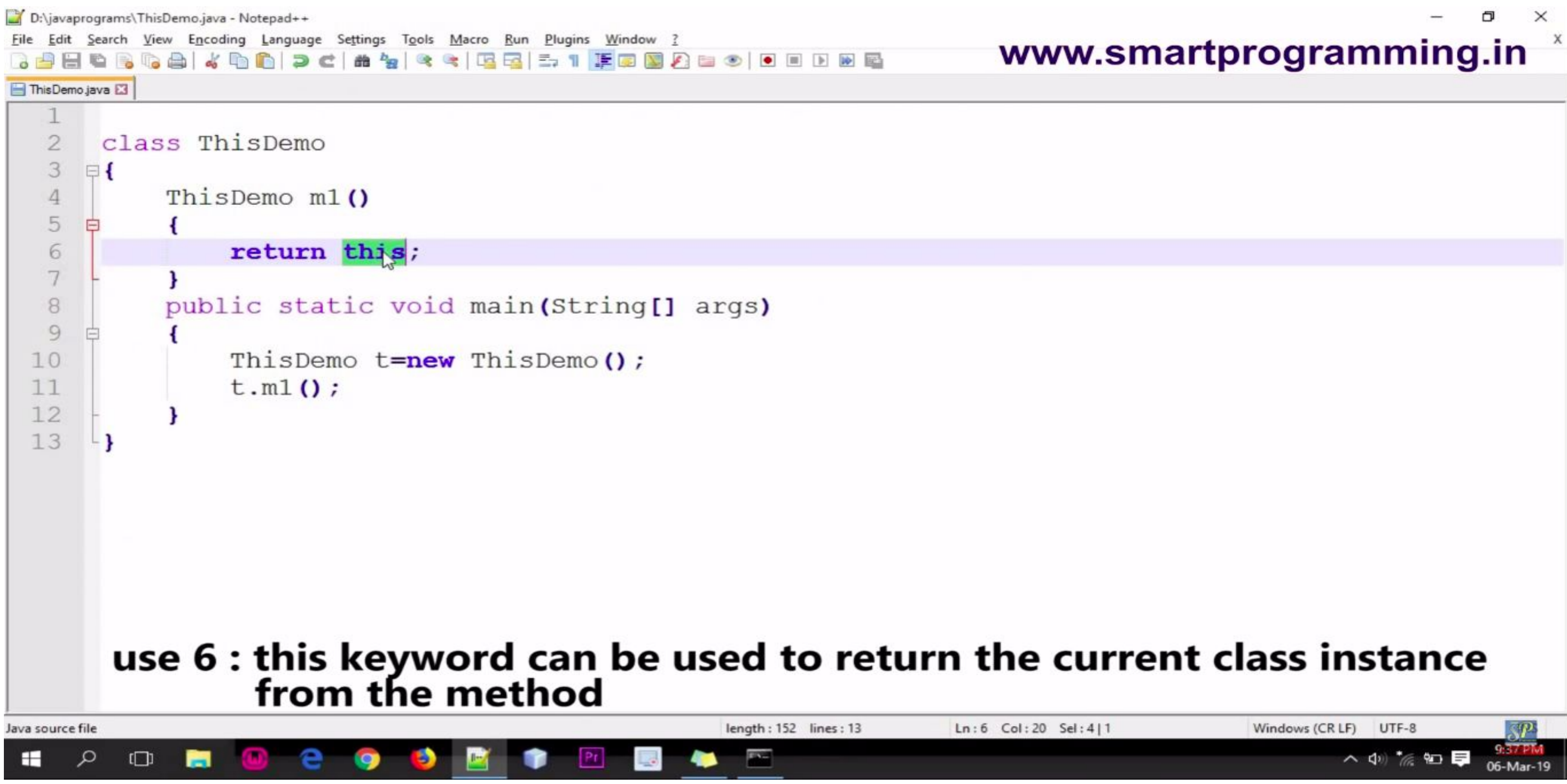


D:\javaprograms\ThisDemo.java - Notepad++

www.smartprogramming.in

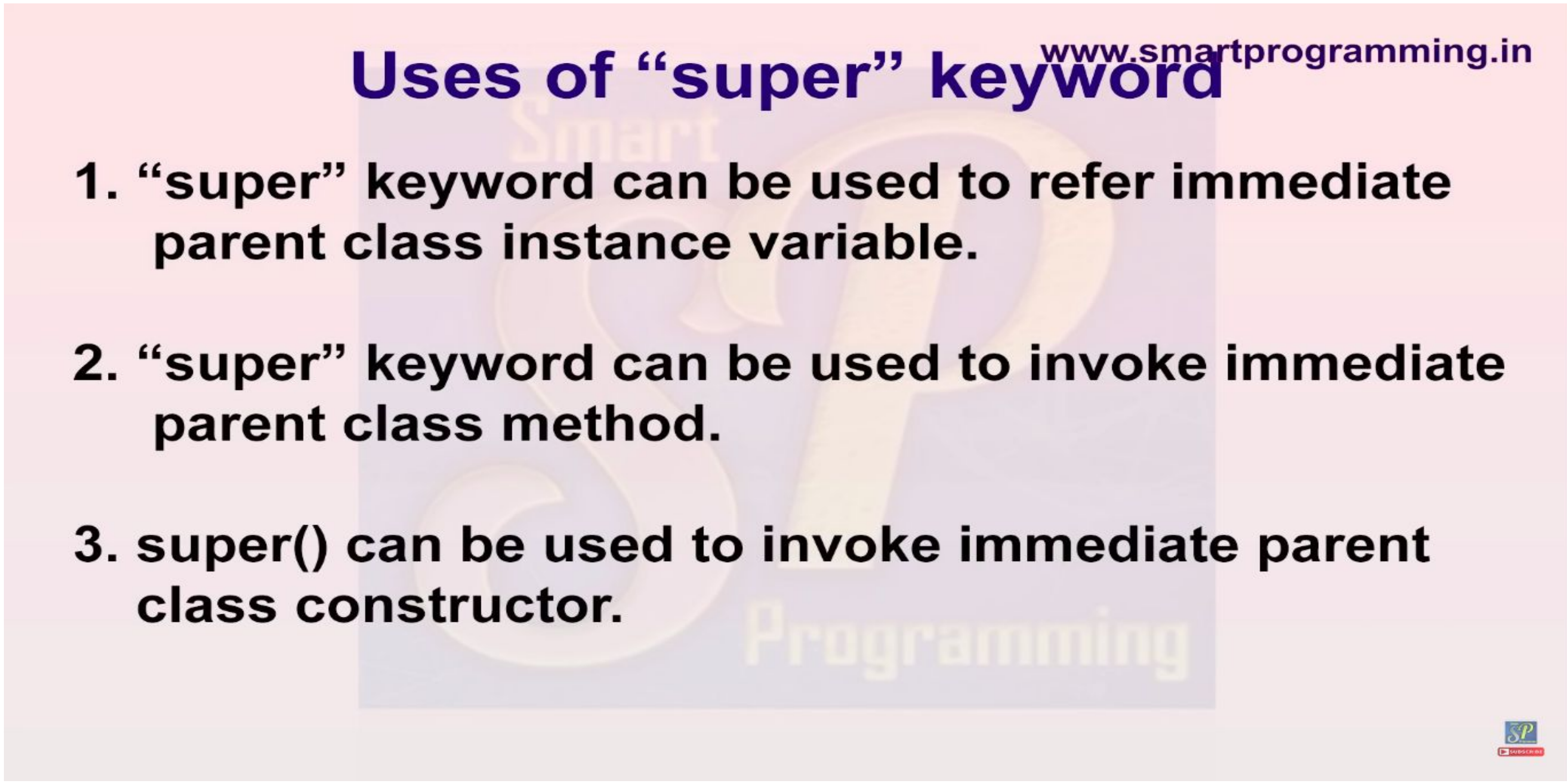
```java
class Test
{
    Test(ThisDemo td)
    {
        System.out.println("test class constructor");
    }
}
class ThisDemo
{
    void m1()
    {
        Test t=new Test(this);
    }
    public static void main(String[] args)
    {
        ThisDemo t=new ThisDemo();
        t.m1();
    }
}
```

use 5 : this keyword can be used to pass as an argument in the constructor call.


D:\javaprograms\ThisDemo.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
www.smartprogramming.in
ThisDemo.java
Java source file
length : 152 lines : 13
Ln : 6 Col : 20 Sel : 4 | 1
Windows (CR LF) UTF-8
9:37 PM
06-Mar-19

```java

class ThisDemo
{
      ThisDemo m1()
      {
            return this;
      }
      public static void main(String[] args)
      {
            ThisDemo t=new ThisDemo();
            t.m1();
      }
}
```

use 6 : this keyword can be used to return the current class instance from the method

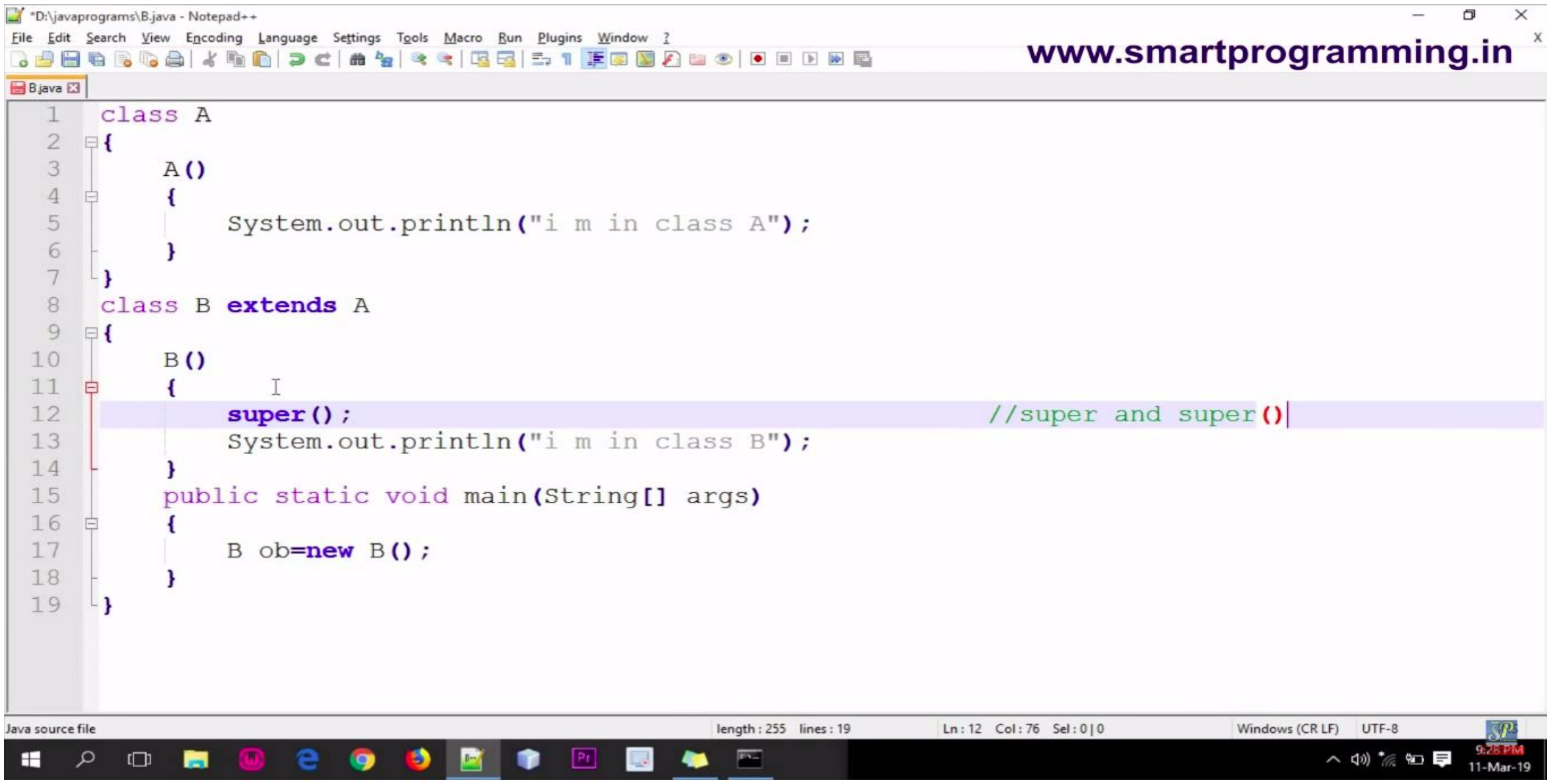

www.smartprogramming.in

Uses of “super” keyword

1. “super” keyword can be used to refer immediate parent class instance variable.

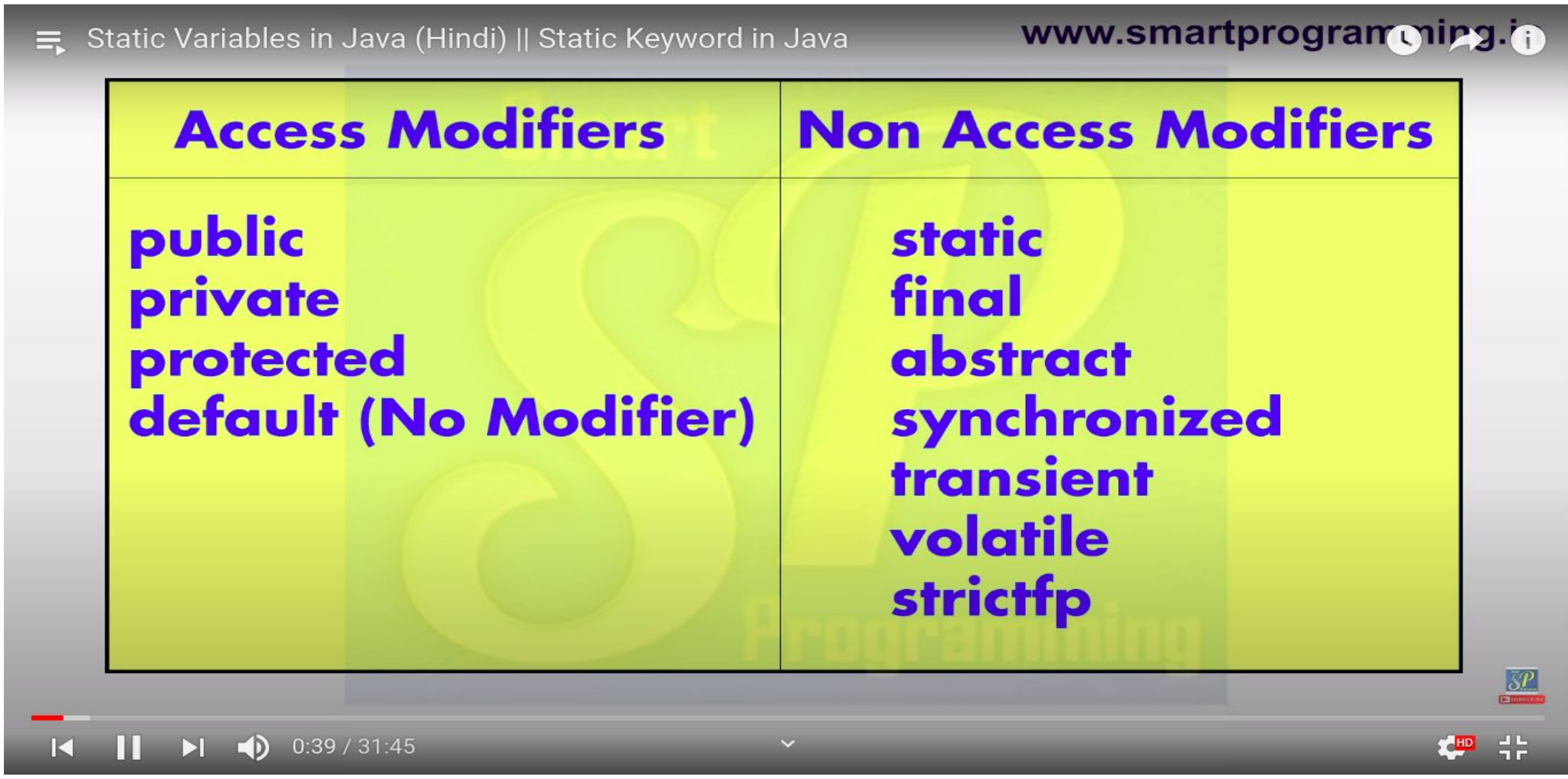
2. “super” keyword can be used to invoke immediate parent class method.

3. super() can be used to invoke immediate parent class constructor.

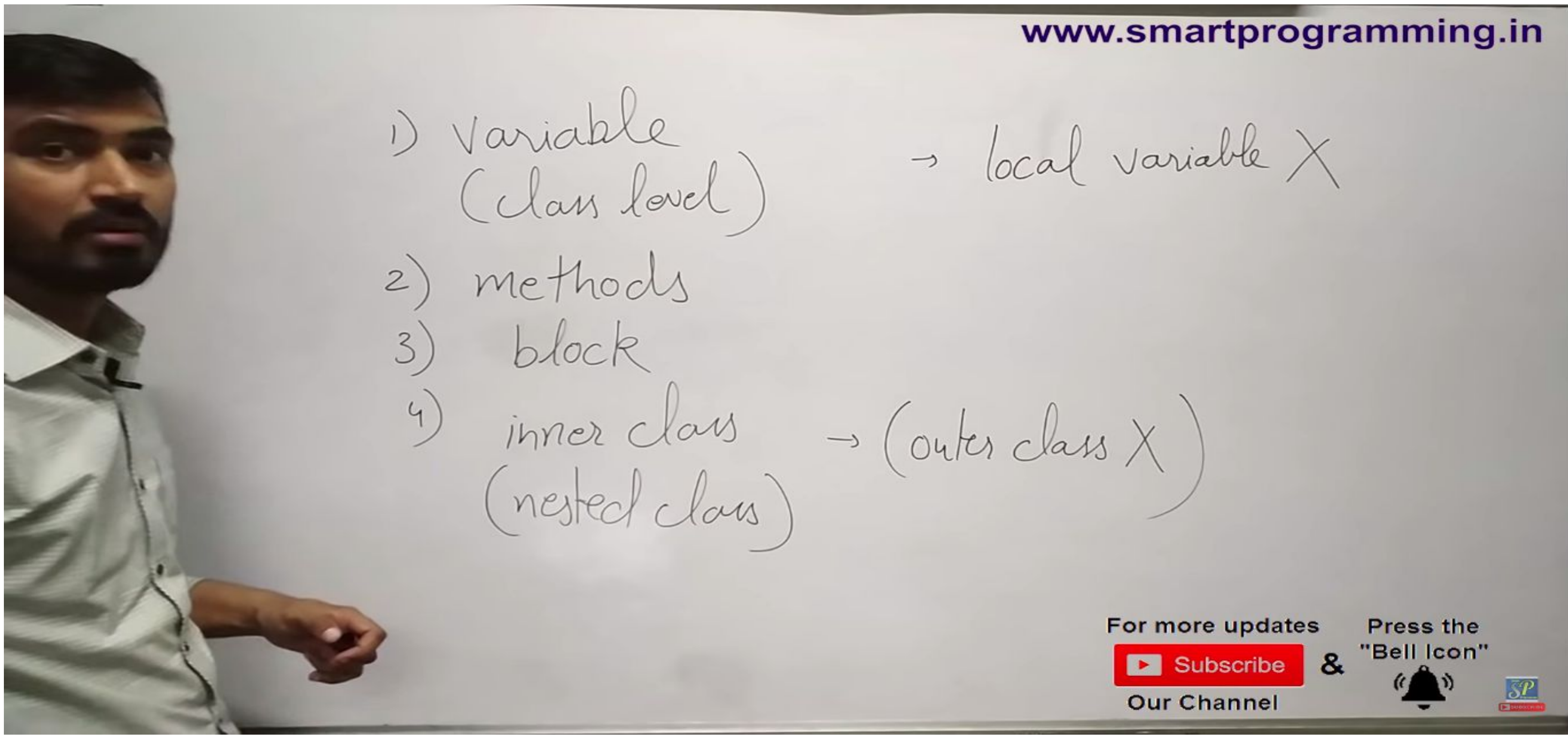

*D:\javaprograms\B.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
www.smartprogramming.in
B.java

```
class A
{
	A()
	{
		System.out.println("i m in class A");
	}
}
class B extends A
{
	B()
	{
		super();                                              //super and super()
		System.out.println("i m in class B");
	}
	public static void main(String[] args)
	{
		B ob=new B();
	}
}
```

Java source file
length : 255 lines : 19
Ln : 12 Col : 76 Sel : 0 | 0
Windows (CR LF) UTF-8
9:28 PM
11-Mar-19



Access Modifiers	Non Access Modifiers
public private protected default (No Modifier)	static final abstract synchronized transient volatile strictfp

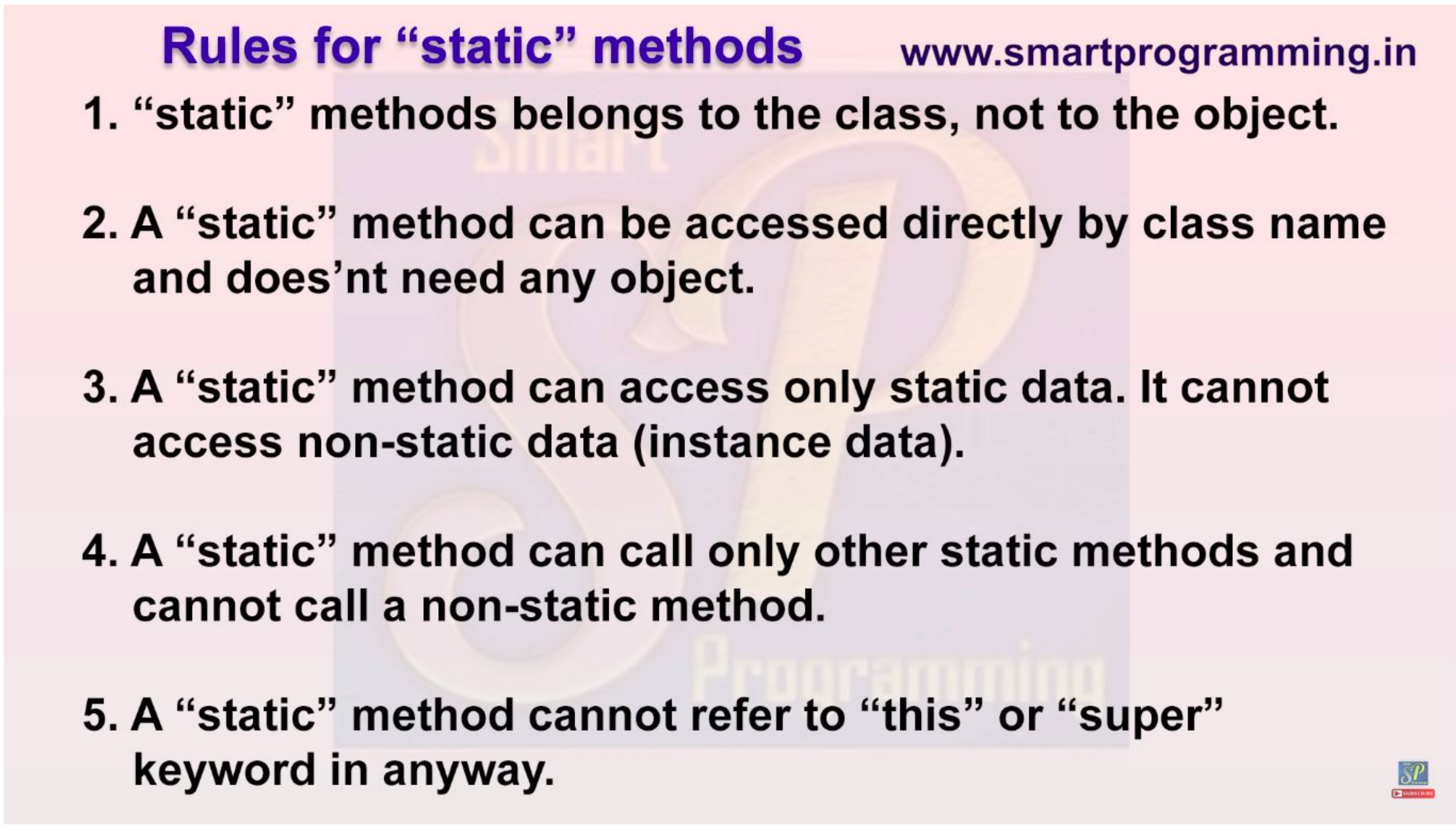

www.smartprogramming.in

1) variable (class level) → local variable X

2) methods

3) block

4) inner class (nested class) → (outer class X)



Rules for “static” methods

www.smartprogramming.in

1. “static” methods belongs to the class, not to the object.
2. A “static” method can be accessed directly by class name and does’nt need any object.
3. A “static” method can access only static data. It cannot access non-static data (instance data).
4. A “static” method can call only other static methods and cannot call a non-static method.
5. A “static” method cannot refer to “this” or “super” keyword in anyway.