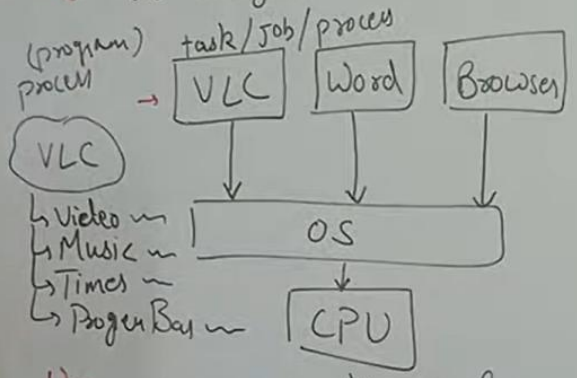
MULTI TASKING

→ performing multiple task

Def. at single time



Use → increases the performance of CPU

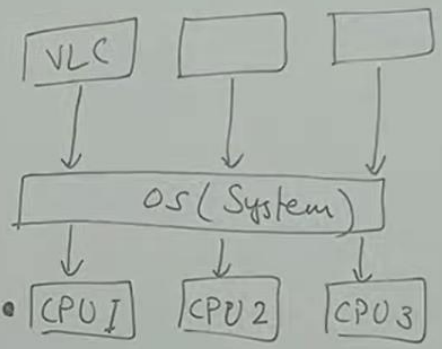
Types → 2 ...:-

1) ...ess based Multitasking (MP)

2) Thread based Multitasking (MT)

MULTI PROCESSING

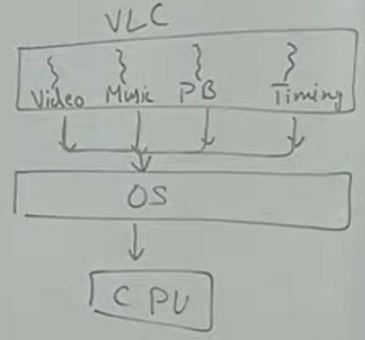
→ When one system is connected to multiple processors in order to complete the task



→ It is best suitable at system level or OS level

MULTI ...

→ Executing multiple ... seacls (sub-proces, small task ... at Single time



→ Softwares
→ Games
→ Animations

Difference between Multitasking, Multiprocessing and Multithreading in Hindi by Deepak

www.smartprogramming.in

→ VLC (process, program)

class VLC
{
  psvm( )
  {
    = playVideo();
    = startMusic();
  }
}

class Video → thread 1
{
  void playVideo()
  {
  }
}

Music → Thread 2

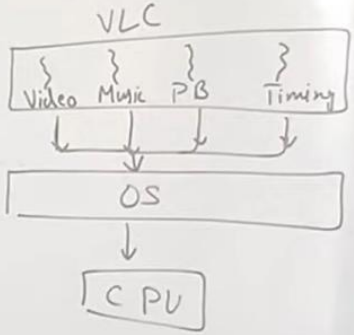
void startMusic()
{
}

→ Multithreading is best suitable at programming level

→ Java provides predefined API for multithreading (10-20%)

- Thread, • Runnable,
- ThreadGroup,
- Concurrency, • ThreadPool

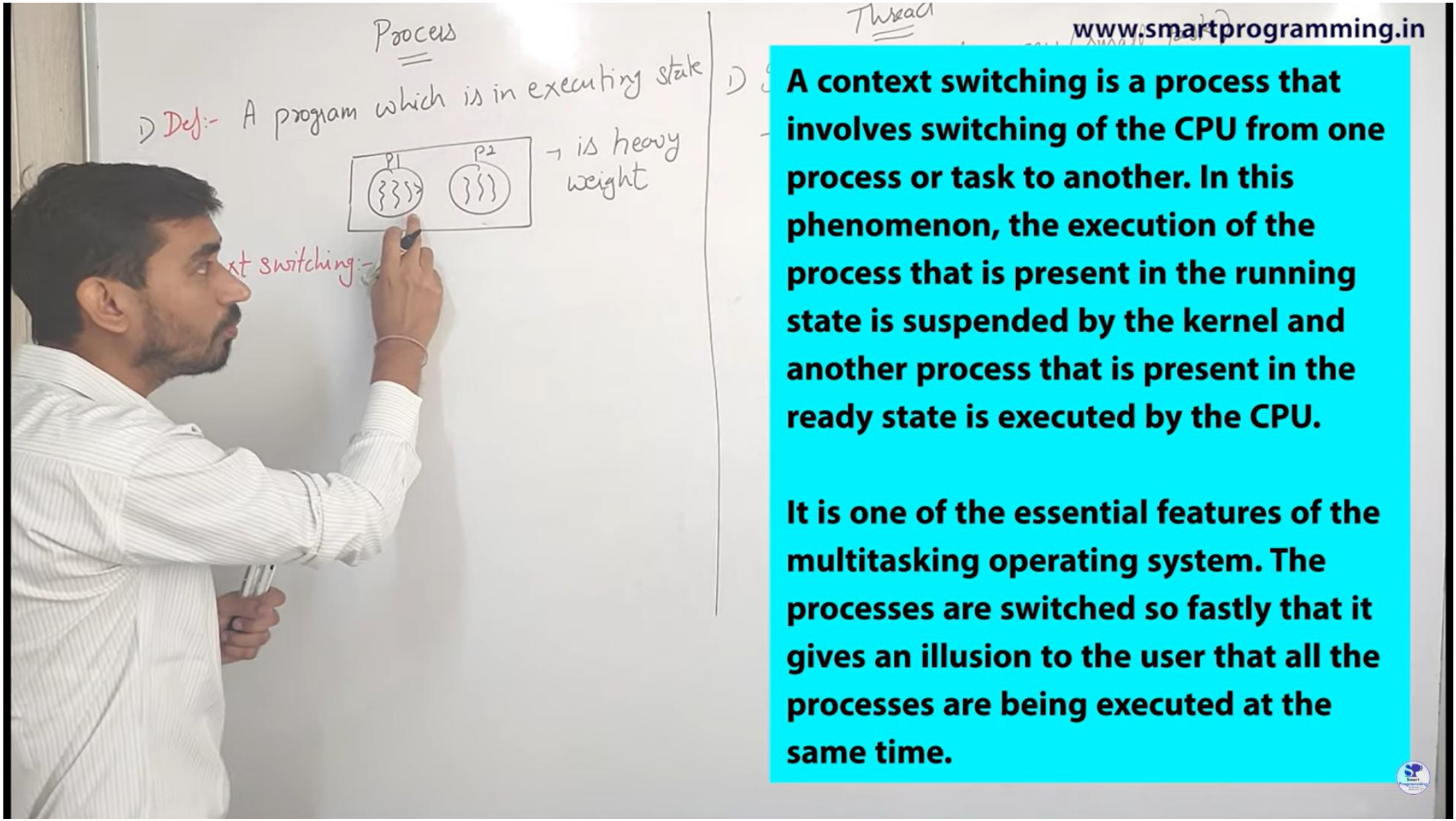
→ Executing multiple tasks (sub-process, small task) at single time



→ Softwares
→ Games
→ Animations

23:04 / 24:04

Scroll for details

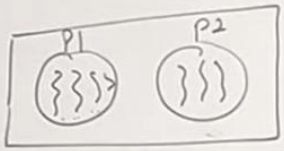


A context switching is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.

It is one of the essential features of the multitasking operating system. The processes are switched so fastly that it gives an illusion to the user that all the processes are being executed at the same time.

Process

1) Def:- A program which is in executing state



→ is heavy weight

2) Context switching:- takes more time

3) Communication:- takes more time

4) Address Space :- each process has different address space

5) Dependency:- Process are not dependent on each other.

6) Synchronization:-

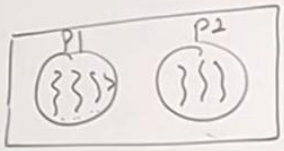
Thread

1) **When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issues. For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file. So in this type of situation we use SYNCHRONIZATION.**

(will deeply explained in further tutorials)

Process

1) Def:- A program which is in executing state



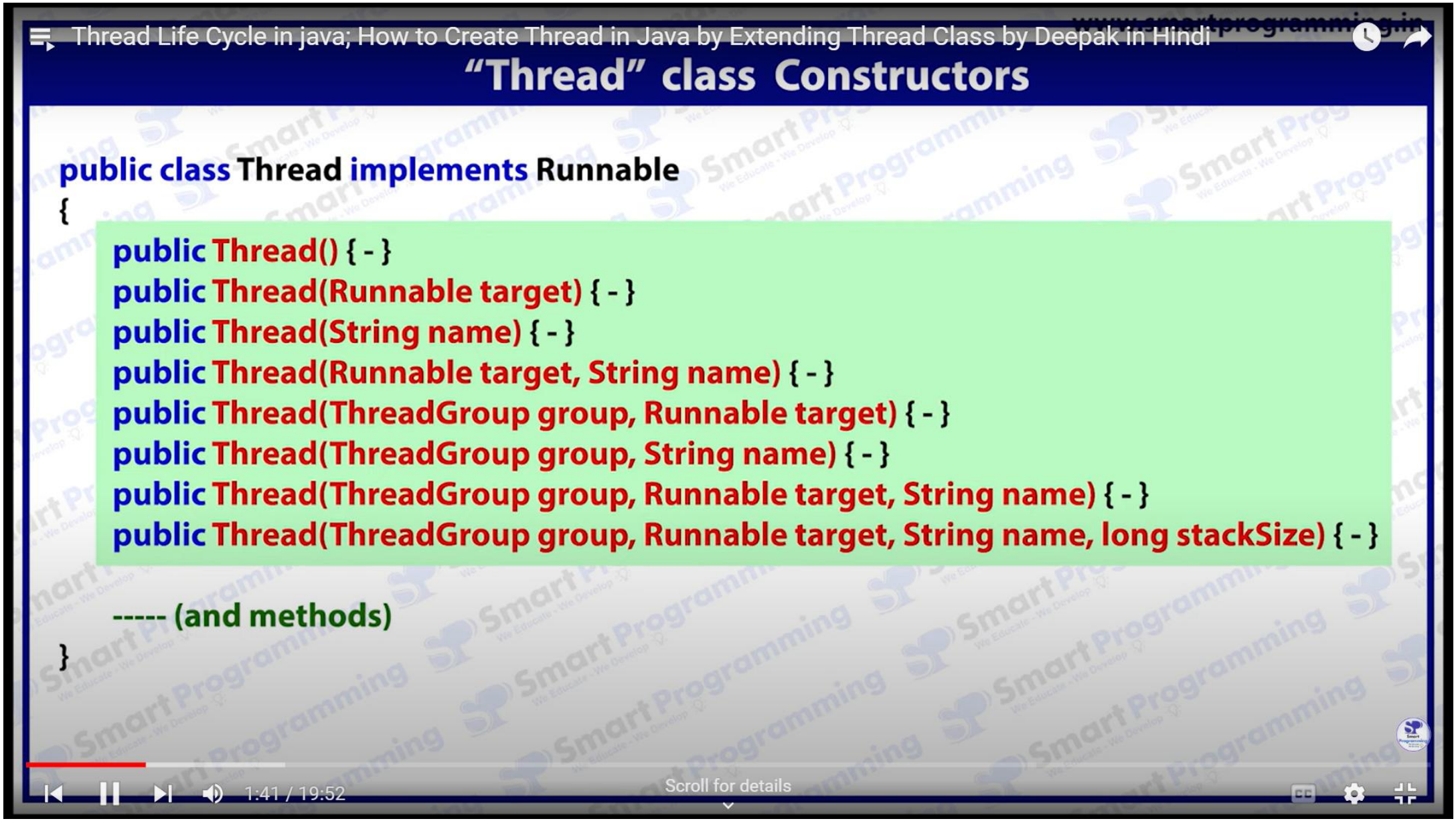
→ is heavy weight

2) Context switching:- takes more time
3) Communication:- takes more time
4) Address Space :- each process has different address space
5) Dependency:- Process are not dependent on each other.
6) Synchronization:- Process does not require Synchronization
7) Resource consuption:- is more
8) Time of creation:- more time
9) Time for termination:- more time

Thread

1) It is subpart of process (small task)
→ light weight

2) takes less time
3) takes less time
4) threads share same address space
5) threads are dependent on each other
6) Threads may require Synchronization
7) less
8) less time.
9) less time


Thread Life Cycle in java; How to Create Thread in Java by Extending Thread Class by Deepak in Hindi
www.smartprogramming.in
"Thread" class Constructors
public class Thread implements Runnable
{
public Thread() { - }
public Thread(Runnable target) { - }
public Thread(String name) { - }
public Thread(Runnable target, String name) { - }
public Thread(ThreadGroup group, Runnable target) { - }
public Thread(ThreadGroup group, String name) { - }
public Thread(ThreadGroup group, Runnable target, String name) { - }
public Thread(ThreadGroup group, Runnable target, String name, long stackSize) { - }
----- (and methods)
}
1:41 / 19:52
Scroll for details

"Thread" class methods (Part 1)

```
public class Thread implements Runnable
{
    public void run() { - }
    public synchronized void start() { - }
    public static native Thread currentThread();
    public final native boolean isAlive();

    public final String getName() { - }
    public final synchronized void setName(String name) { - }

    public final boolean isDaemon() { - }
    public final void setDaemon(boolean on) { - }

    public final int getPriority() { - }
    public final void setPriority(int newPriority) { - }

    ----- (many more methods)
}
```

→ Basic Methods

→ Naming Methods

→ Daemon Thread Methods

→ Priority Based Methods

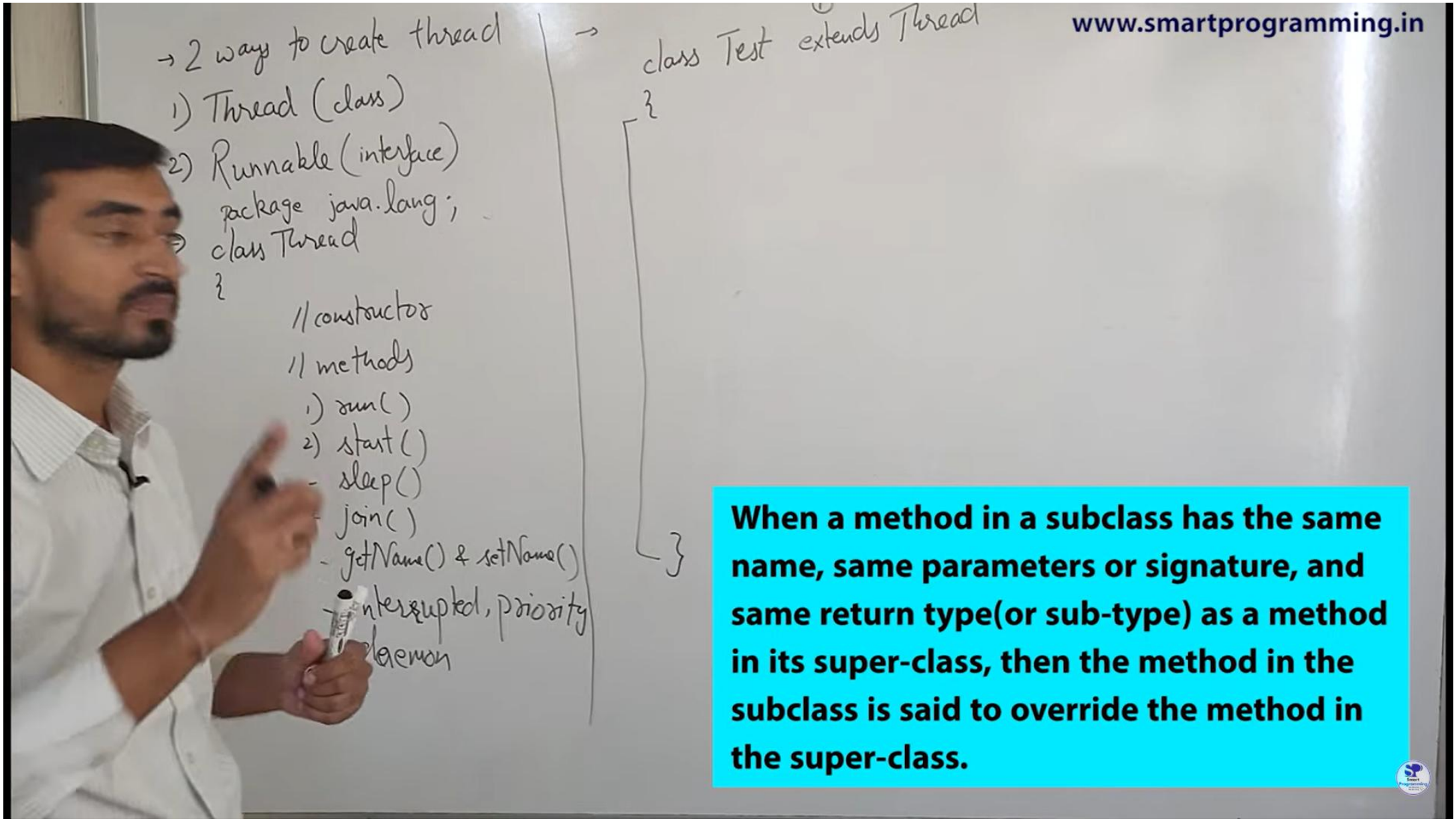
"Thread" class methods (Part 2)

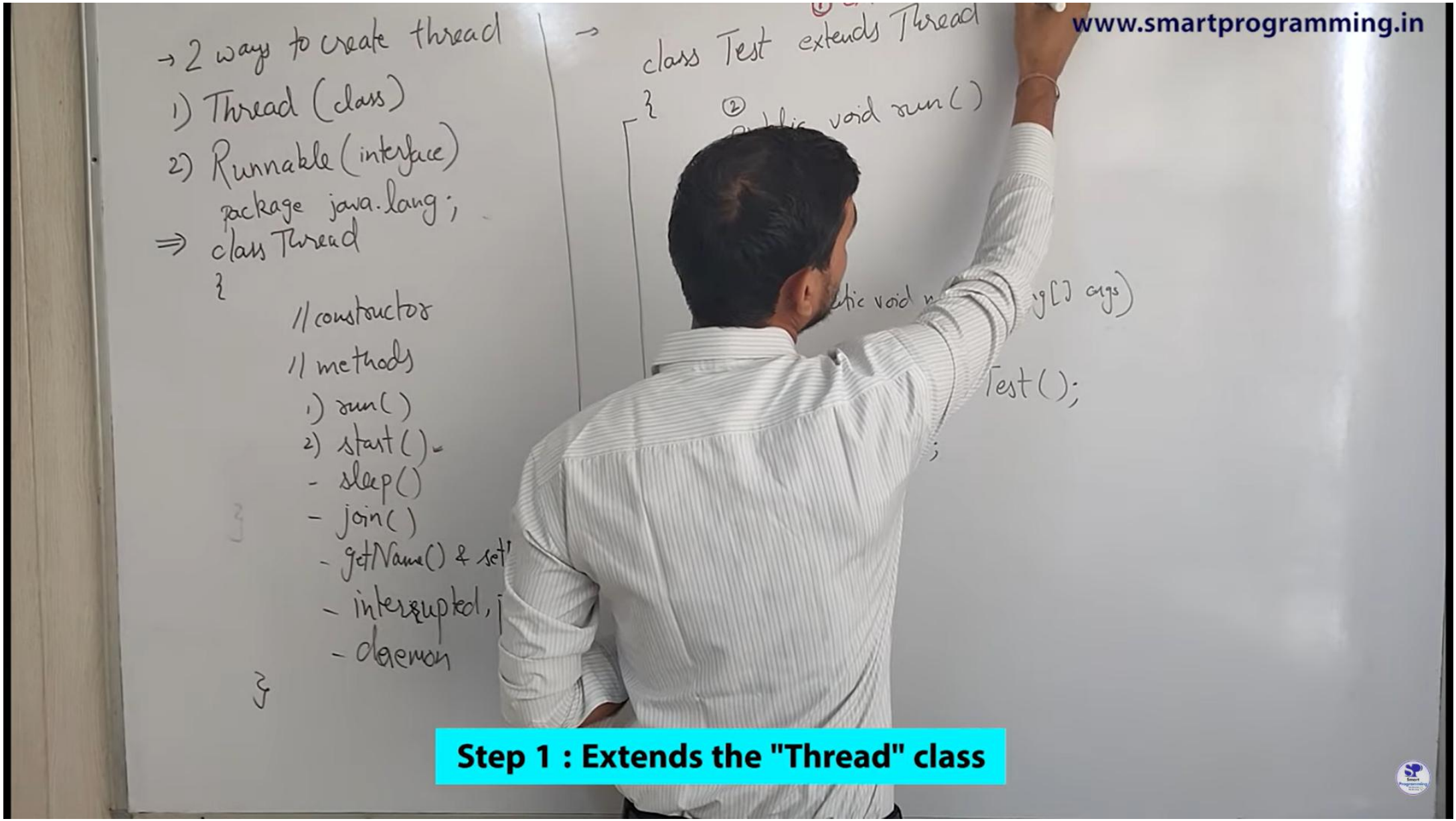
```
public class Thread implements Runnable
{
      ----- (many more methods)

      public static native void sleep(long millis) throws InterruptedException;
      public static native void yield();
      public final void join() throws InterruptedException { - }
      public final void suspend() { - }
      public final void resume() { - }
      public final void stop() { - }
      public void destroy() { - }

      public void interrupt() { - }
      public boolean isInterrupted() { - }
      public static boolean interrupted() { - }
}
```

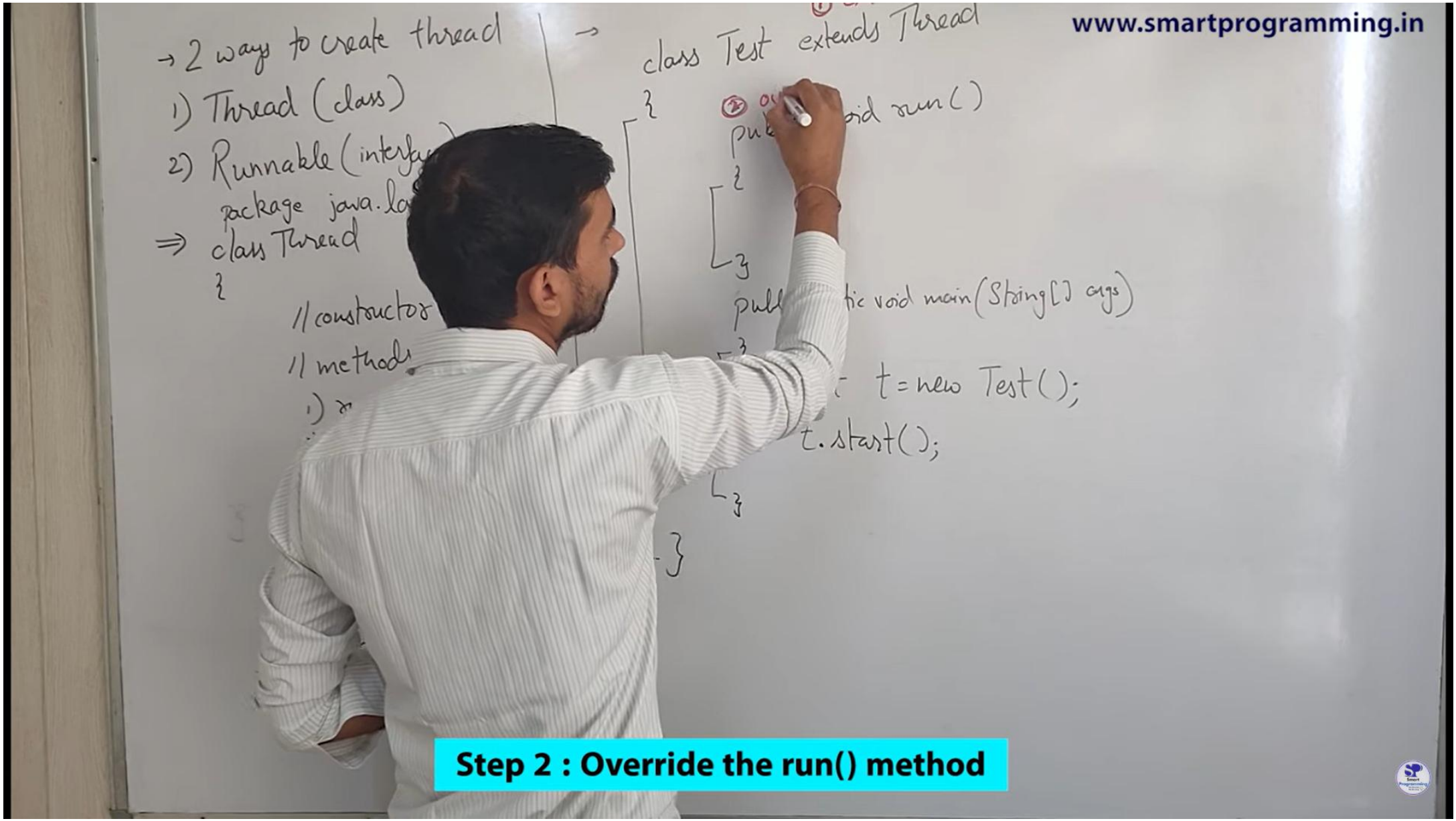
→ Prevent Thread Execution Methods

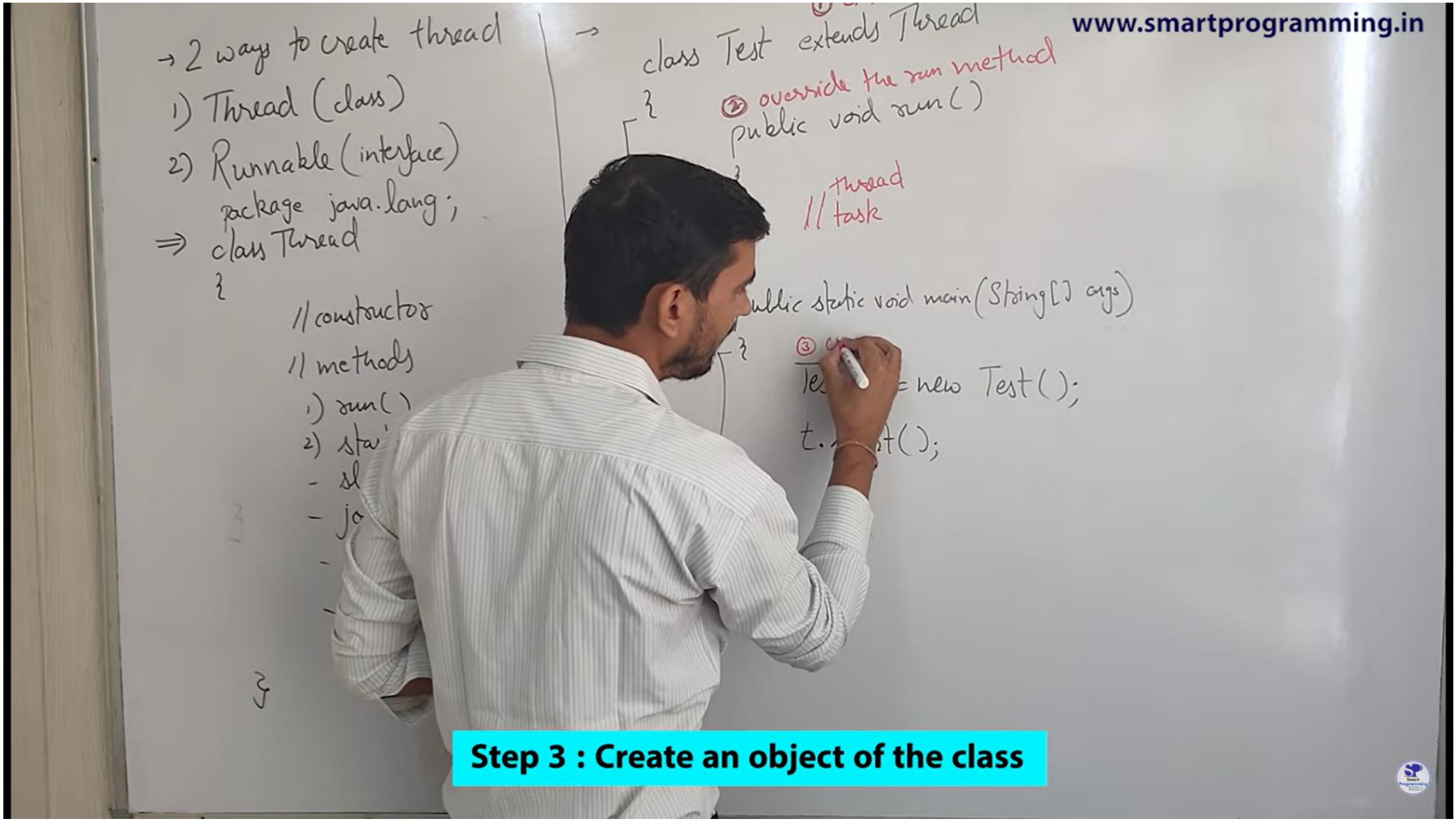
→ Depricated methods

→ Interrupting a thread Methods


→ 2 ways to create thread
1) Thread (class)
2) Runnable (interface)
package java.lang;
class Thread
{
//constructor
//methods
1) run()
2) start()
- sleep()
- join()
- getName() & setName()
- interrupted, priority
- daemon
→
class Test extends Thread
{
}
www.smartprogramming.in
When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

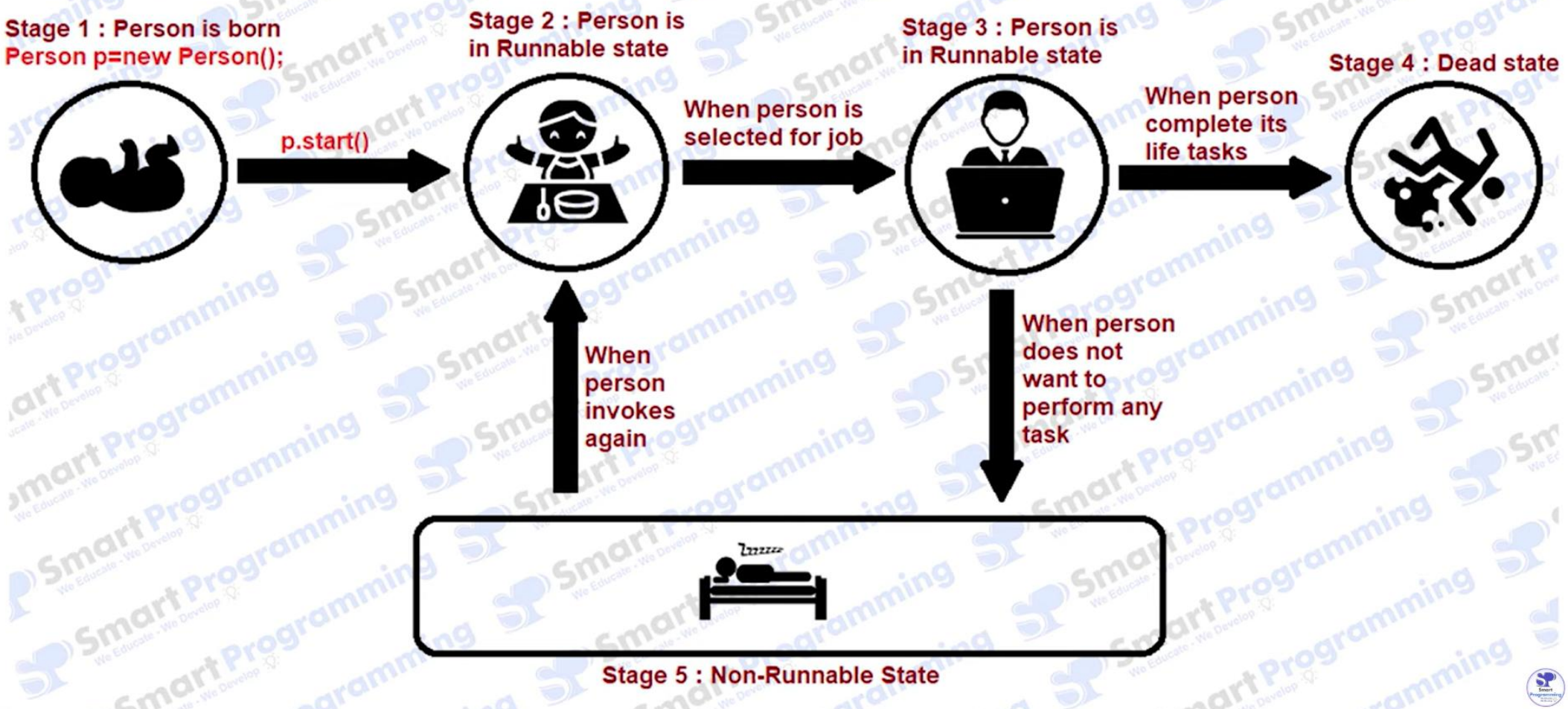

→ 2 ways to create thread
1) Thread (class)
2) Runnable (interface)
package java.lang;
⇒ class Thread
{
// constructor
// methods
1) run()
2) start()
- sleep()
- join()
- getName() & set
- interrupted, i
- daemon
}
→
class Test extends Thread
{
② void run()
Test();
www.smartprogramming.in
Step 1 : Extends the "Thread" class

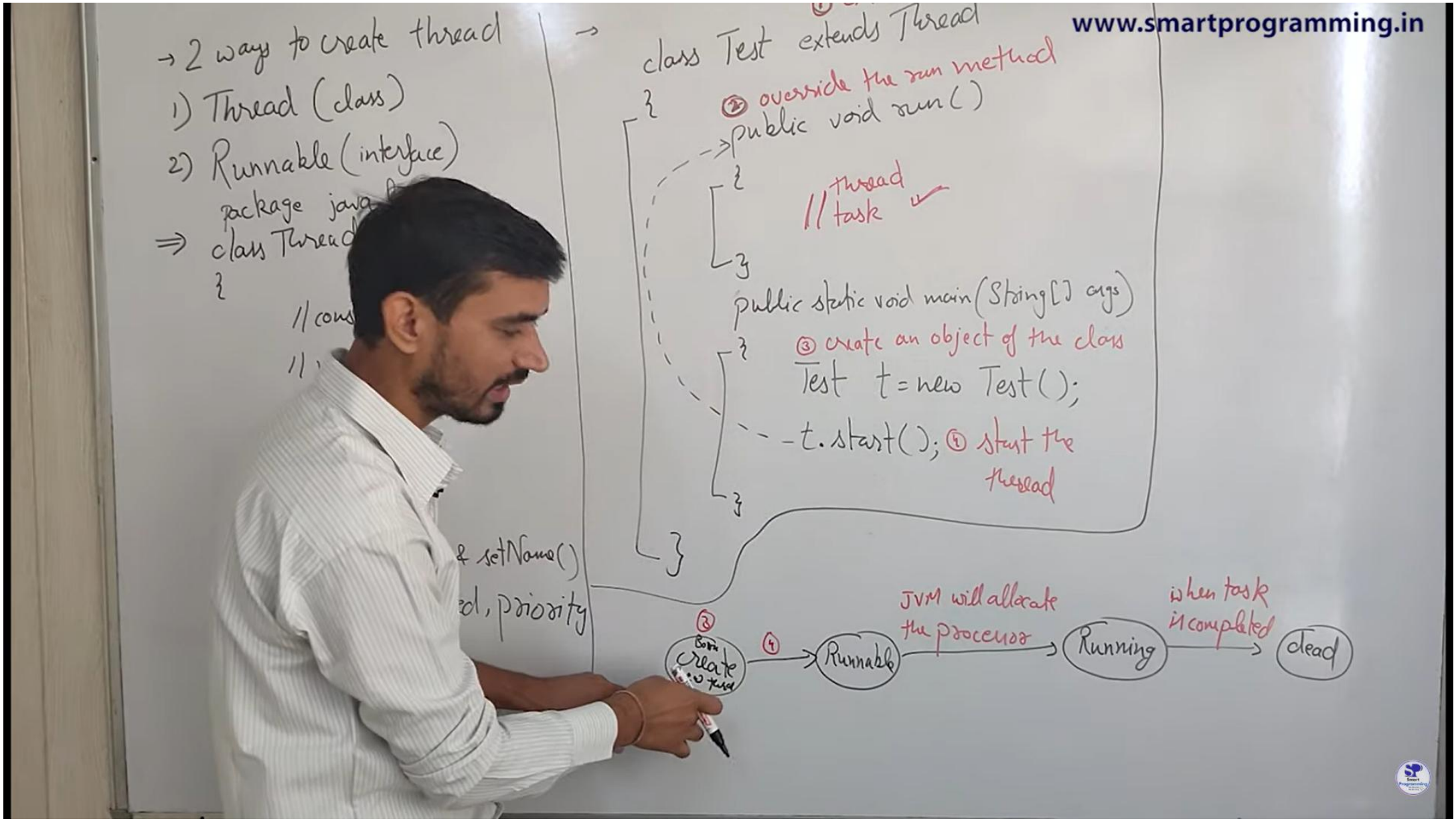

→ 2 ways to create thread
1) Thread (class)
2) Runnable (interface)
package java.lang
⇒ class Thread
{
//constructor
//methods
1) →
→
class Test extends Thread
{
② public void run()
{
}
public static void main(String[] args)
t = new Test();
t.start();
}
}
www.smartprogramming.in
Step 2 : Override the run() method


→ 2 ways to create thread
1) Thread (class)
2) Runnable (interface)
package java.lang;
⇒ class Thread
{
//constructor
//methods
1) run()
2) sta
- sl
- jo
}
→
class Test extends Thread
{
② override the run method
public void run()
{
//thread task
}
public static void main(String[] args)
{
③
Te = new Test();
t.
t();
}
www.smartprogramming.in
Step 3 : Create an object of the class


www.smartprogramming.in
→ 2 ways to create thread
1) Thread (class)
2) Runnable (interface)
package java.lang;
⇒ class Thread
{
// constructor
// methods
1) run()
2) start()
- sleep()
- join()
- getName() & set
- interrupted
- daemon
}
→
class Test extends Thread
{
② override the run method
public void run()
{
}
}
n(String[] args)
bject of the class
= new Test();
st(); ① start the
Step 4 : Start the thread by using start() method

Assume Person is Thread




www.smartprogramming.in
→ 2 ways to create thread
1) Thread (class)
2) Runnable (interface)
package java
⇒ class Thread
{
// cons
//
& setName()
ed, priority
→
class Test extends Thread
{
② override the run method
public void run()
{
// thread task
}
public static void main(String[] args)
{
③ create an object of the class
Test t = new Test();
t.start(); ④ start the thread
}
}
③
④
Runnable
JVM will allocate the processor
Running
when task is completed
dead

→ 2 ways to create thread

1) Thread (class)
2) Runnable (interface)

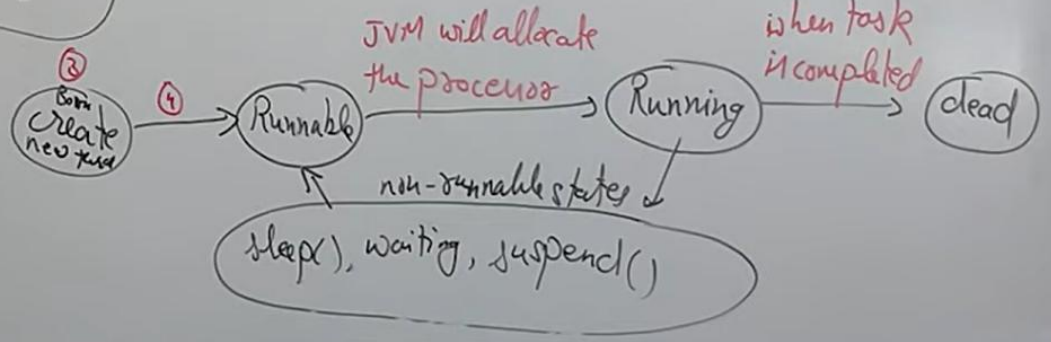
package java.lang;

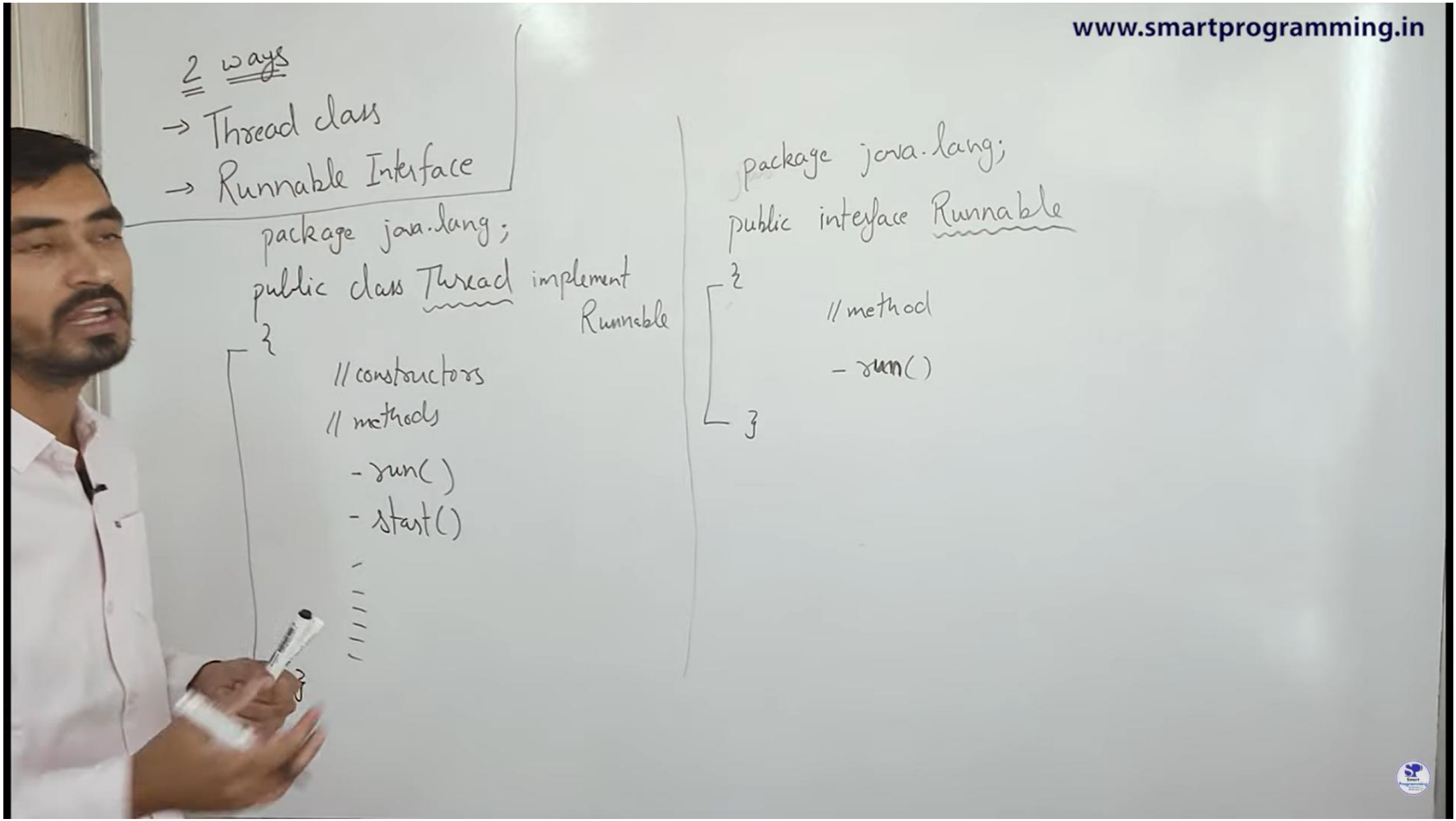
⇒ class Thread
{
    // constructor
    // methods
    1) run()
    2) start()
    - sleep()
    - join()
    - getName() & setName()
    - interrupted, priority
    - daemon
}

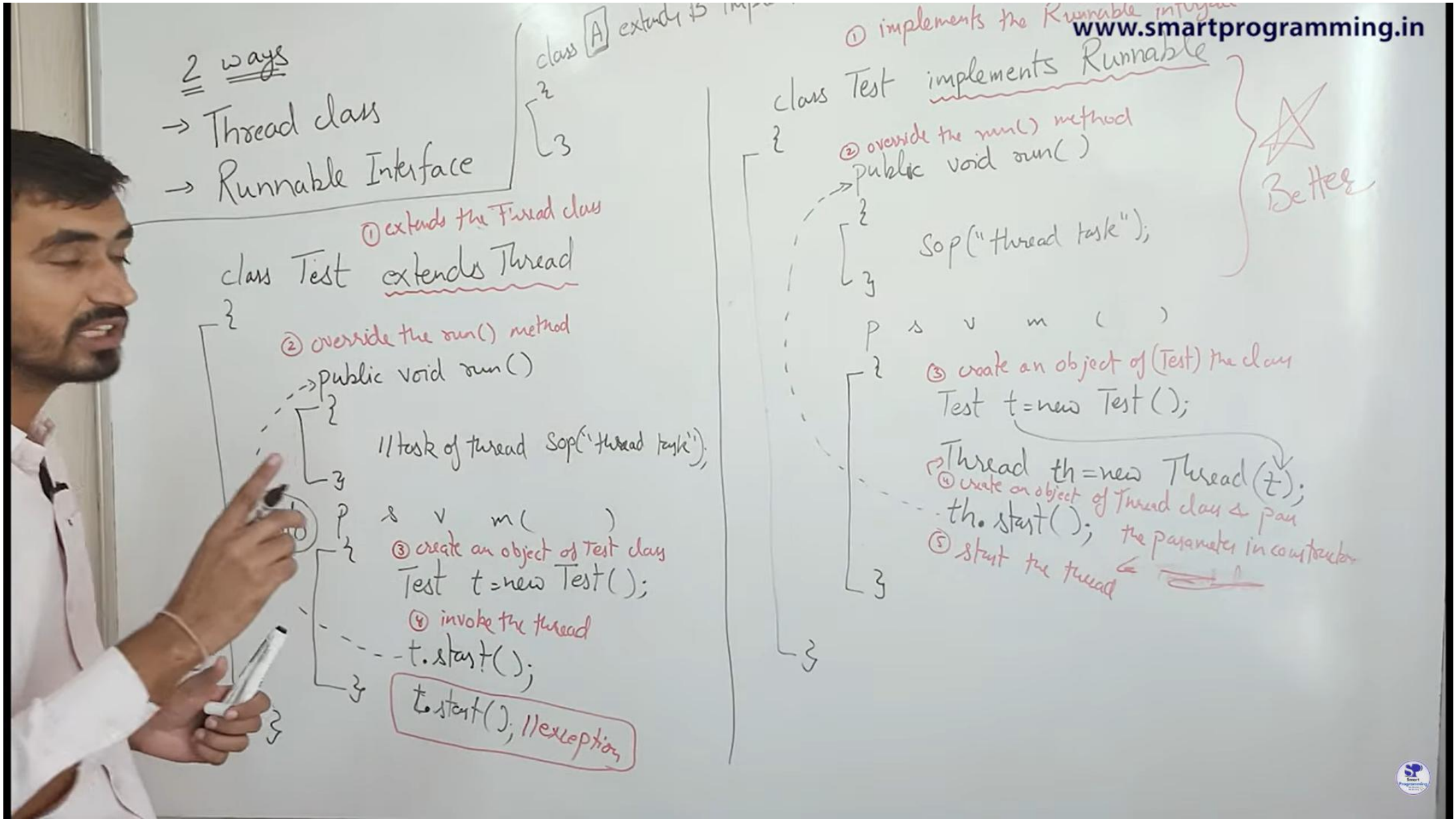
→

class Test extends Thread
{
    ② override the run method
    public void run()
    {
        // thread task
    }
    public static void main(String[] args)
    {
        ③ create an object of the class
        Test t = new Test();
        t.start(); ④ start the thread
    }
}

LIFE CYCLE OF THREAD

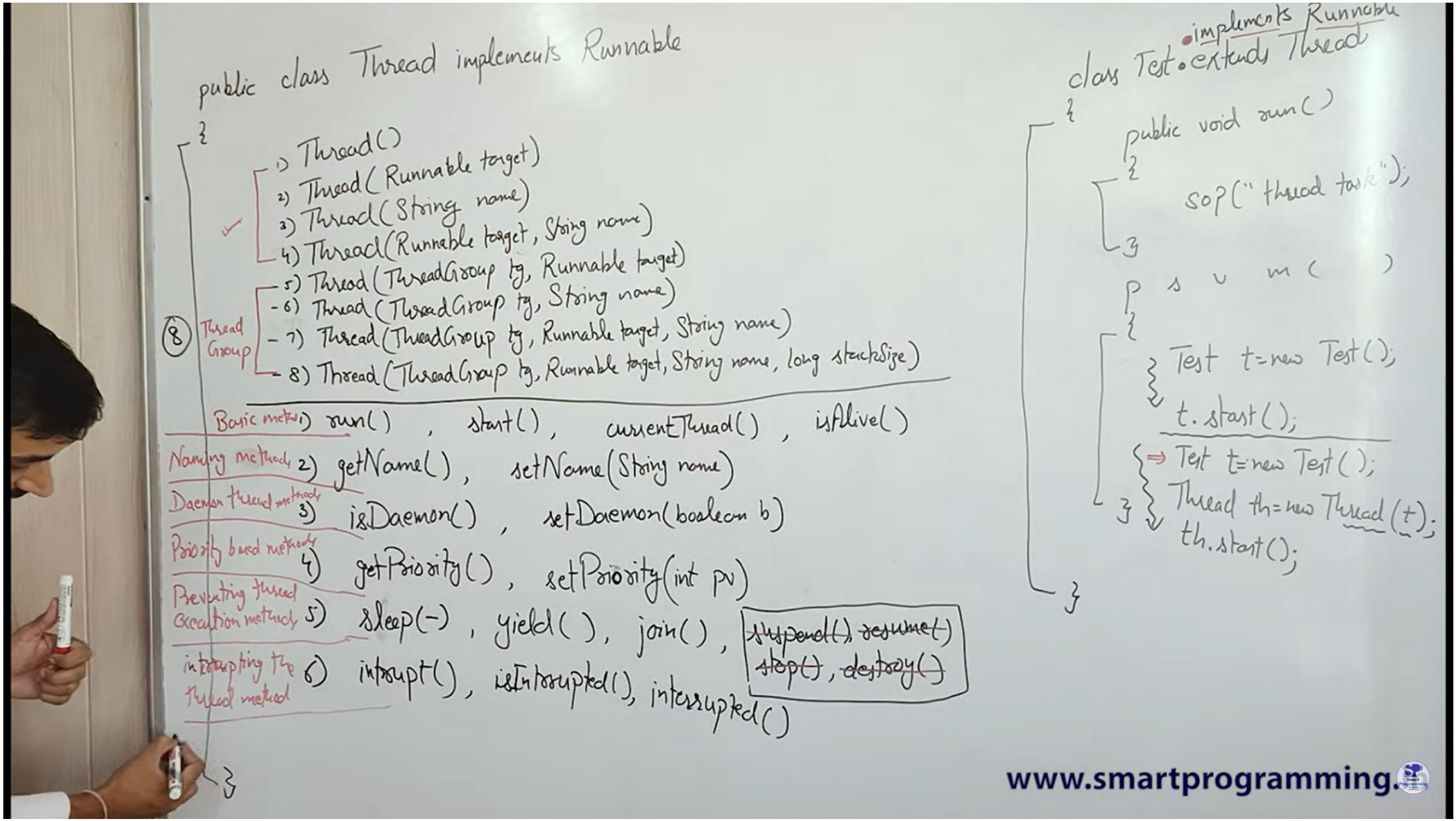





www.smartprogramming.in
2 ways
→ Thread class
→ Runnable Interface
class A extends
① extends the Thread class
class Test extends Thread
{
② override the run() method
public void run()
{
//task of thread Sop("thread task");
}
p s v m( )
{
③ create an object of Test class
Test t=new Test();
④ invoke the thread
t.start();
}
}
t.start(); //exception
① implements the Runnable
class Test implements Runnable
{
② override the run() method
public void run()
{
Sop("thread task");
}
p s v m( )
{
③ create an object of (Test) the class
Test t=new Test();
Thread th=new Thread(t);
④ create an object of Thread class & pass the parameter in constructor
th.start();
⑤ start the thread
}
}
Better

Topics covered in Multithreading till now...!!

1. What is Multitasking, Multiprocessing & Multithreading
2. Difference between Process & Thread
3. Thread Life Cycle & How to create Threads in Java
4. Two different ways to create Threads :-
    1. By extending "Thread" class
    2. By Implementing "Runnable" interface
5. Thread execution cases :-
    1. Performing single task from single thread
    2. Performing single task from multiple threads
    3. Performing multiple task from single thread
    4. Performing multiple task from multiple threads



"Thread" class Constructors

```
public class Thread implements Runnable
{
    public Thread() { - }
    public Thread(Runnable target) { - }
    public Thread(String name) { - }
    public Thread(Runnable target, String name) { - }
    public Thread(ThreadGroup group, Runnable target) { - }
    public Thread(ThreadGroup group, String name) { - }
    public Thread(ThreadGroup group, Runnable target, String name) { - }
    public Thread(ThreadGroup group, Runnable target, String name, long stackSize) { - }

    ----- (and methods)
}
```

"Thread" class methods (Part 1)

```
public class Thread implements Runnable
{
    public void run() { - }
    public synchronized void start() { - }
    public static native Thread currentThread();
    public final native boolean isAlive();

    public final String getName() { - }
    public final synchronized void setName(String name) { - }

    public final boolean isDaemon() { - }
    public final void setDaemon(boolean on) { - }

    public final int getPriority() { - }
    public final void setPriority(int newPriority) { - }

    ----- (many more methods)
}
```

Basic Methods

Naming Methods

Daemon Thread Methods

Priority Based Methods

"Thread" class methods (Part 2)

```
public class Thread implements Runnable
{
      ----- (many more methods)

      public static native void sleep(long millis) throws InterruptedException;
      public static native void yield();
      public final void join() throws InterruptedException { - }
      public final void suspend() { - }
      public final void resume() { - }
      public final void stop() { - }
      public void destroy() { - }

      public void interrupt() { - }
      public boolean isInterrupted() { - }
      public static boolean interrupted() { - }
}
```

→ Prevent Thread Execution Methods

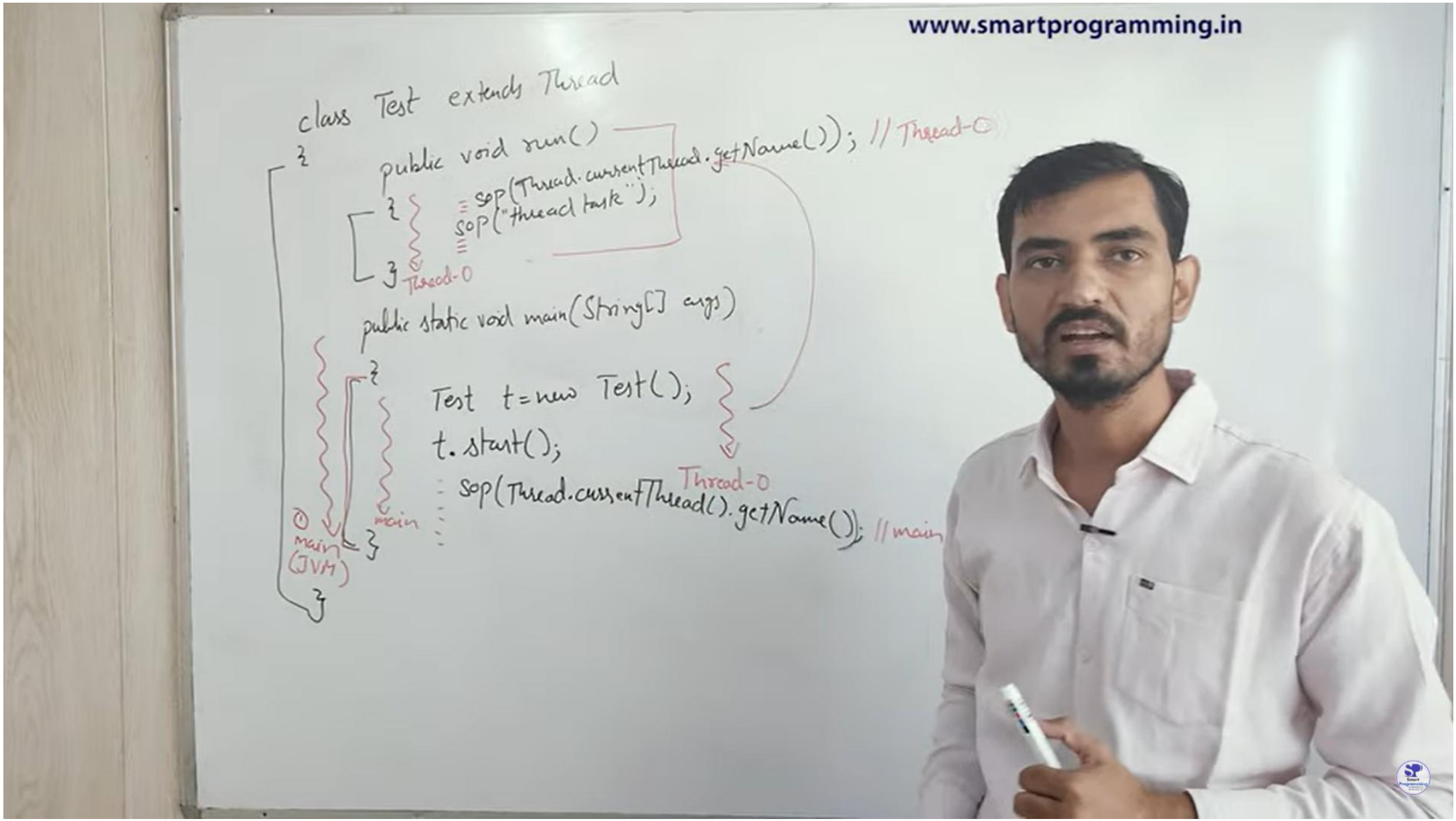
→ Depricated methods

→ Interrupting a thread Methods

"Thread" class methods (Part 3)

```
public class Object
{
    public final void wait() throws InterruptedException { - }
    public final native void notify();
    public final native void notifyAll();
}
```

Inter-Thread Communication Methods



1. Def:- which run in the background of another thread
2. Use:- It provides service to the threads
3. Eg:- Garbage Collector, finalizer, Attach listener, Signal dispatcher etc
4. Methods:- a. public final void setDaemon(boolean b)
   b. public final boolean isDaemon()
5. Program:-

```
class Test extends Thread
{
    public void run()
    {
        sop("child thread");
    }
    p s v m( )
    {
        sop("Main thread");
        Test t=new Test();
        t.setDaemon(true);
        t.start();
    }
}
```

Thread-0 (Daemon thread)

main

service

Daemon

main Daemon

Daemon ← By default Daemon

6. Life:- Its life depends on another thread
7. Daemon nature:- It inherits the nature/properties from its parent ~~child~~ thread.
8) JVM role in Daemon thread

Daemon

Conc 1:- we have to create daemon thread before starting the thread, If we create daemon thread after starting it, it will throw run-time exception ie. IllegalThreadStateException

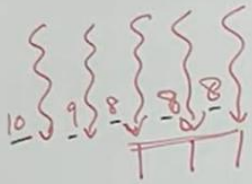
Conc 2:- We cannot create main thread as daemon thread

Thread.currentThread().setDaemon(true);

9. Most of the times, Daemon threads has low priority, but we can change its priority according to our needs

www.smartprogramming.in

① ⇒ JVM provides the priorities to each thread & according to these priorities, JVM allocates the processor



② Priorities are represented in the form of integer value which ranges from 1-10

- 1 → MIN_PRIORITY
  5 → NORM_PRIORITY
  10 → MAX_PRIORITY
- Below are not the priorities
  → 0 , <1 , >10
  → MINIMUM_PRIORITY
  → LOW_PRIORITY
  → MIDIUM_PRIORITY,
  → NORMAL_PRIORITY
  → MAXIMUM_PRIORITY
  → HIGH_PRIORITY

③ Methods:-
→ public final void setPriority(int value)
→ public final int getPriority()

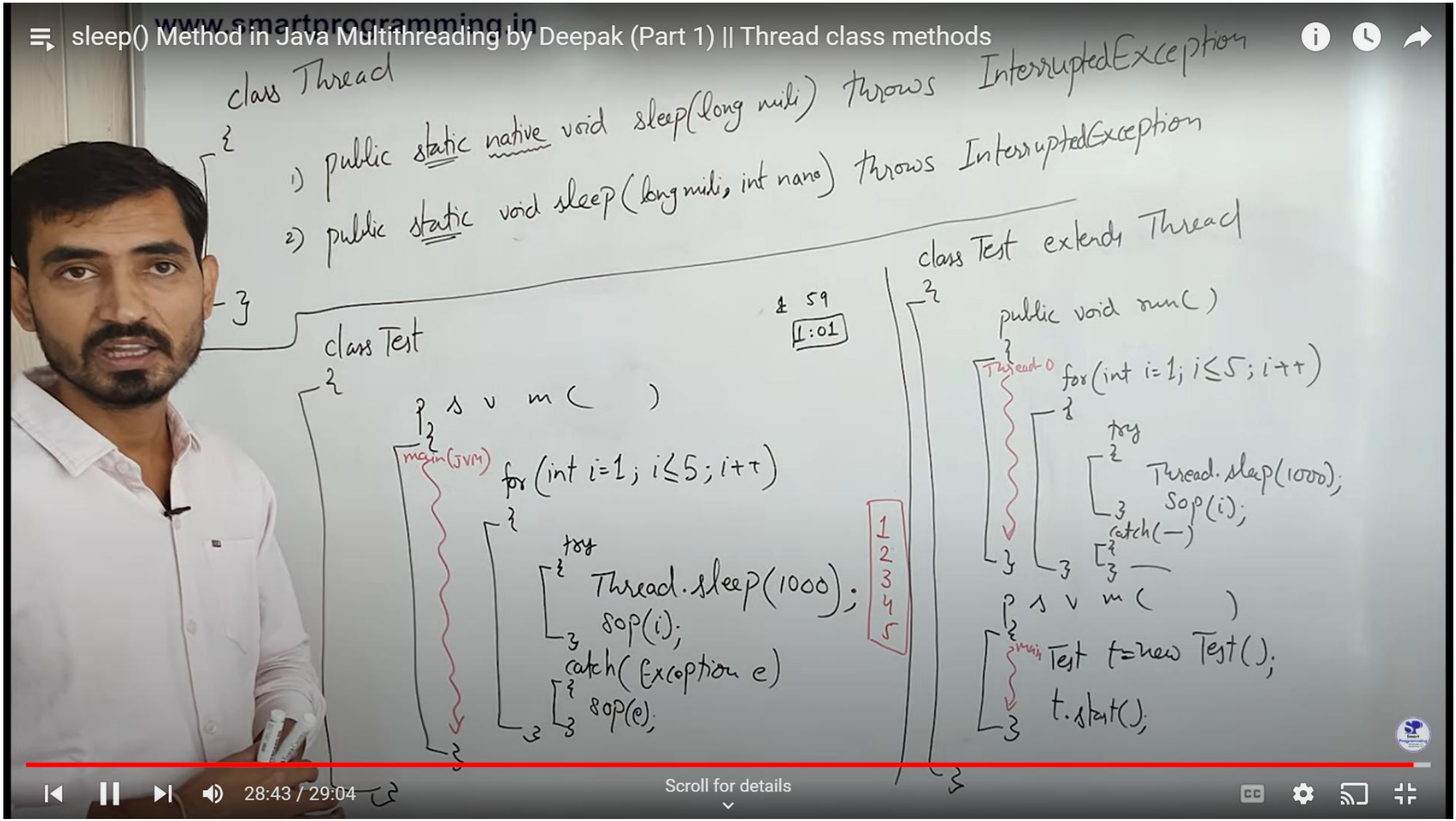
```
class Test extends Thread
{
    public void run()
    {
        sop("child thread");
        sop(Thread.currentThread().getPriority());
    }
}
psvm()
{
    sop(Thread.currentThread().getPriority());
    Test t=new Test();
    t.start();
    t.setPriority(10);
}
```

Thread-0 (5)
main (5)

① Nature Default :- Priorities are inherited from parent thread
④ By default main thread priority is 5
⑤ If priority value is not in b/w 1-10, then it will throw runtime exception i.e. IllegalArgumentException

sleep() method important points

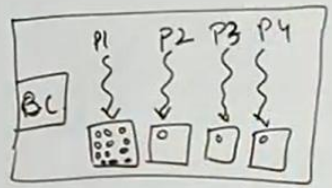
1. If the value of milliseconds is negative then "IllegalArgumentException" is thrown.
2. If the value of nanoseconds is not in the range 0-999999 then "IllegalArgumentException" is thrown.
3. Whenever we want to use the sleep() method we also need to handle the "InterruptedException". If we will not handle it, the JVM will show a compilation error.
4. When any thread is sleeping and if any other thread interrupts it, then it throws "InterruptedException".
5. The sleep() method always pauses the current thread execution. When the JVM finds the sleep() method in code, it checks that which thread is running and pause the execution of thread.
6. When we use sleep() method to pause the execution of thread. the thread scheduler assigns the CPU to another thread if any thread exists. So, there is no guarantee that the thread wakes up exactly after the time specified in sleep() method. It totally depends on the thread scheduler.
7. While the thread is sleeping, it doesn't lose any locks or monitors that it had acquired before sleeping.


www.smartprogramming.in
sleep() Method in Java Multithreading by Deepak (Part 1) || Thread class methods
class Thread
1) public static native void sleep(long mili) throws InterruptedException
2) public static void sleep(long mili, int nano) throws InterruptedException
class Test
psvm( )
main(JVM)
for(int i=1; i<=5; i++)
try
Thread.sleep(1000);
sop(i);
catch(Exception e)
sop(e);
1 2 3 4 5
class Test extends Thread
public void run( )
Thread-0
for(int i=1; i<=5; i++)
try
Thread.sleep(1000);
sop(i);
catch(—)
psvm( )
main
Test t=new Test( );
t.start( );
28:43 / 29:04
Scroll for details

www.smartprogramming.in

① yield():- which stops the current executing thread & give a chance to other threads for execution

eg. ⇒



② Working:- Java 5:- internally it uses sleep()

Java 6:- Thread provides the hint to the thread-scheduler, then it depends on thread-scheduler to accept or ignore the hint.

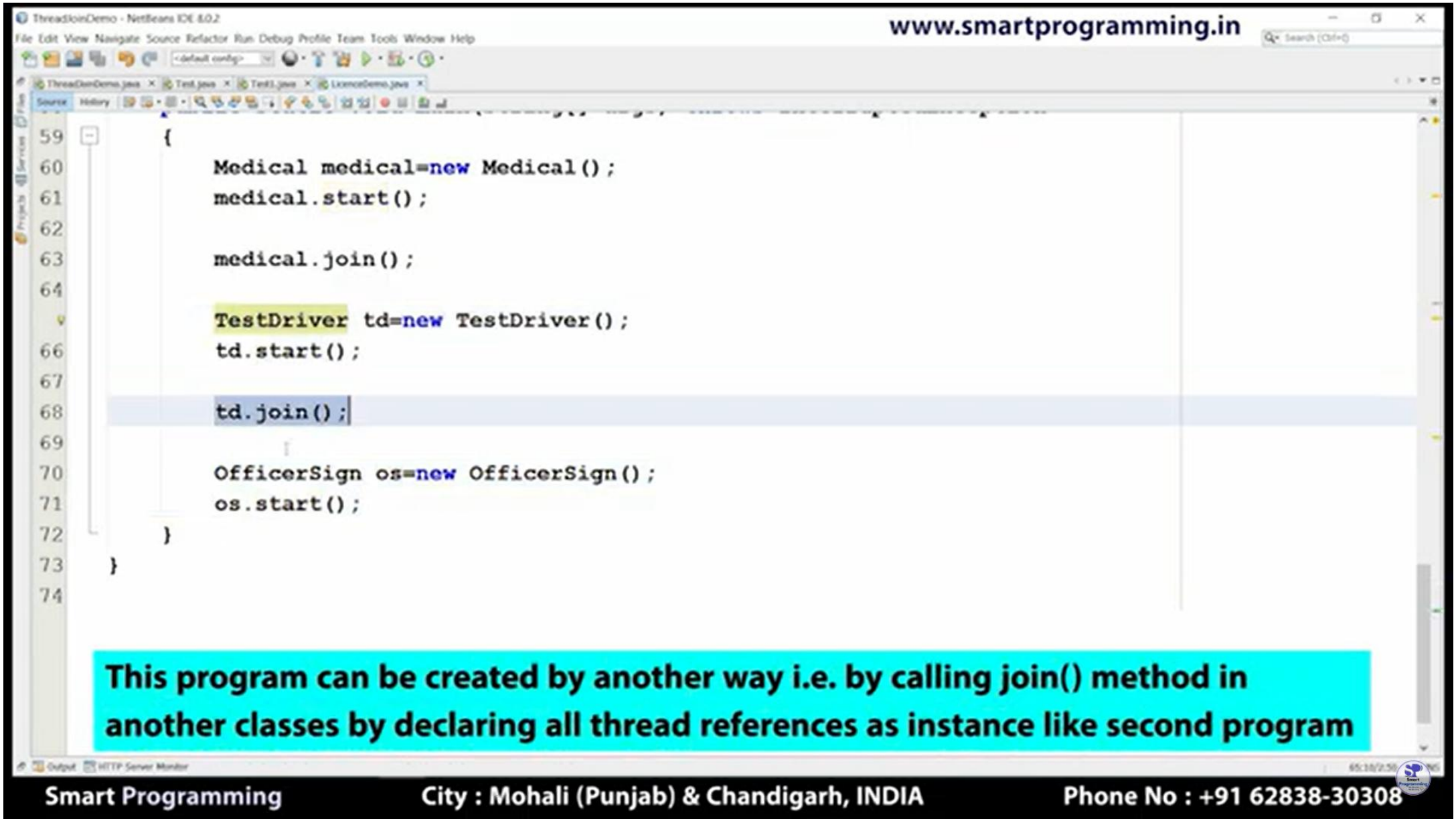
③ Method:- public static native void yield();

④ Output may (vary) different

```
class Test extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            Sop(Thread.currentThread().
                getName()+" -"+i);
        }
    }
    p s v m(  )
    {
        Test t=new Test();
        t.start();
        for(int i=1; i<=5; i++)
        {
            Thread.yield();
            Sop("main thread"+i);
        }
    }
}
```

Thread-0

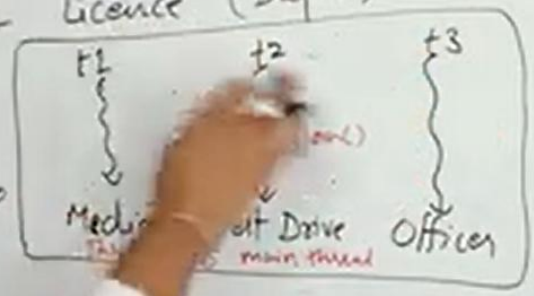
main


ThreadJoinDemo - NetBeans IDE 8.0.2
www.smartprogramming.in
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
{
Medical medical=new Medical();
medical.start();
medical.join();
TestDriver td=new TestDriver();
td.start();
td.join();
OfficerSign os=new OfficerSign();
os.start();
}
}
This program can be created by another way i.e. by calling join() method in another classes by declaring all thread references as instance like second program

www.smartprogramming.in

① join() :- If a thread wants to wait for another thread to complete its task, then we should use join() method.

② Eg - Licence (Deepak)



③ Meth... a. public final void join() throws ...

... final synchronized void join(long ms) ,,

public final synchronized void join(long ms, int ns)

```
class Test extends Thread
{
  public void run()
  {
    try
    {
      for(int i=1; i<=5; i++)
      {
        sop("child thread"+i)
        Thread.sleep(1000);
      }
    }
    catch(--){--}
  }
  P s v m(--)
  {
    Test t=new Test();
    t.start(); t.join();
    try
    {
      for(int i=1; i<=5; i++)
      {
        sop("main thread"+i);
        Thread.sleep(1000);
      }
    }
    catch(--){--}
  }
```

sleep(), yield() & join() methods prototype

sleep() method Prototype :-

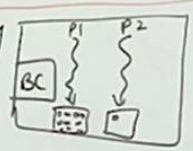
1. public static native void sleep(long ms) throws InterruptedException;
2. public static void sleep(long ms, int ns) throws InterruptedException;

yield() method Prototype :-

1. public static native void yield();

join() method Prototype :-

1. public final void join() throws InterruptedException;
2. public final synchronized void join(long ms) throws InterruptedException;
3. public final synchronized void join(long ms, int ns) throws InterruptedException;

property	sleep()	yield()	join()
1) purpose	If any thread does not want to perform any operation for particular time	It stops the current executing thread & provide chance to another threads of same or higher priority to execute	If a thread wants to wait for another thread to complete its task
2) Examples	1) Timer, ppt, blinking bulbs	1) Shopping 	1) Licence Dept.
3) How the thread invokes again?	1) automatically after provided time period. 2) If thread is interrupted.	1. Automatically invoked by thread-scheduler	1. Automatically invokes after completion of another thread task. 2. After completion of time period. 3. If thread is interrupted.
4) Methods:-	1) sleep(long ms) 2) sleep(long ms, int ns)	1) yield()	1) join() 2) join(long ms) 3) join(long ms, int ns)
5) Is method overloaded?	Yes	No	Yes
6) Exception	Yes(InterruptedException)	No.	Yes(InterruptedException)
7) Is method final	No	No	No
8) Is method static	Yes	Yes	No
9) Is method native	1) native 2) non-native	Yes	No

⇒ These all 3 methods temporary stops the thread execution

1. Use :- It is used to interrupt an executing thread.

2. interrupt() method will only works when the thread is in sleeping or waiting state (sleep() or wait())

3. If a thread is not in sleeping or waiting state then calling an interrupt() method will perform normal behavior.

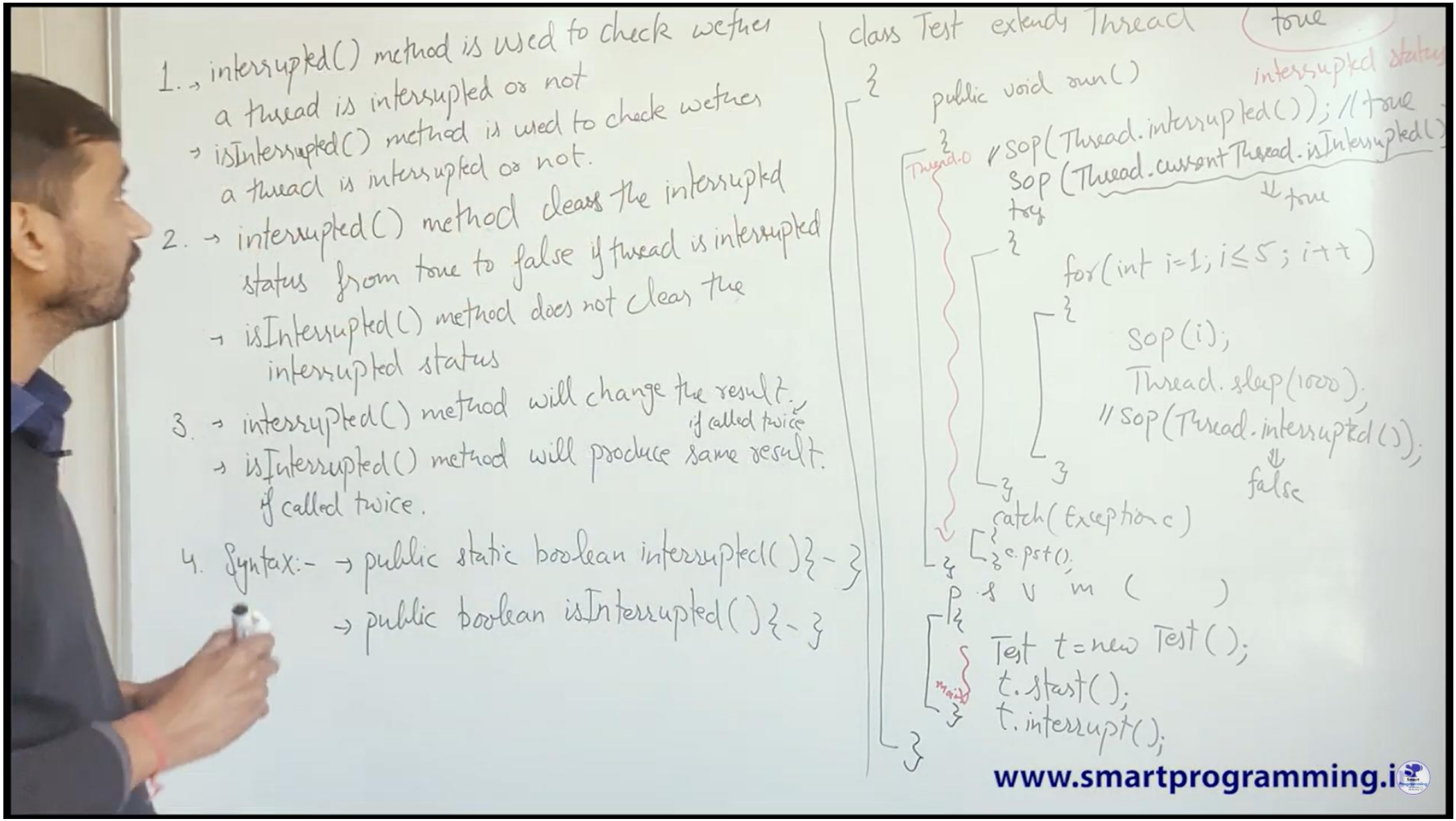
4. When we use an interrupt() method it throws an exception
   InterruptedException

5. Syntax :- public void interrupt(){ - }

```
class Test extends Thread
{
    public void run()
    {
        try
        {
            for(int i=1; i<=5; i++)
            {
                sop(i);
                Thread.sleep(1000);
            }
        }
        catch(Exception e)
        {
            e.pst();
        }
    }
    p s v m( )
    {
        Test t=new Test();
        t.start();
        t.interrupt();
    }
}
```

Thread-0 toy

main


1. → interrupted() method is used to check wether a thread is interrupted or not
→ isInterrupted() method is used to check wether a thread is interrupted or not.
2. → interrupted() method clears the interrupted status from true to false if thread is interrupted
→ isInterrupted() method does not clear the interrupted status
3. → interrupted() method will change the result. if called twice
→ isInterrupted() method will produce same result. if called twice.
4. Syntax:- → public static boolean interrupted(){-}
→ public boolean isInterrupted(){-}
class Test extends Thread
public void run()
Thread-0
// sop(Thread.interrupted()); // true
sop(Thread.currentThread.isInterrupted());
true
try
for(int i=1; i<=5; i++)
sop(i);
Thread.sleep(1000);
// sop(Thread.interrupted());
false
catch(Exception e)
e.pst();
p s v m ( )
Test t=new Test();
t.start();
t.interrupt();
main
true
interrupted status
www.smartprogramming.i

Synchronization Introduction

1. What is Synchronization ?

It is the process by which we control the accessability of multiple threads to a particular shared resource

2. Problem which can occur without synchronization :-

1. Data Inconsistancy
2. Thread interference

3. Advantages of Synchronization :-

1. No data inconsistancy problem
2. No thread interference

4. Disadvantages of Synchronization :-

1. Increases the waiting time period of threads
2. Create performance problems

To overcome synchronization disadvantages, java provides one package i.e. java.util.concurrent

How to achieve Synchronization

Types of Synchronization

2 types of synchronization

Process Synchronization

(Not present in java multithreading)

Thread Synchronization

2 types of thread synchronization

Mutual Exclusive

Can be achieved by 3 ways :-

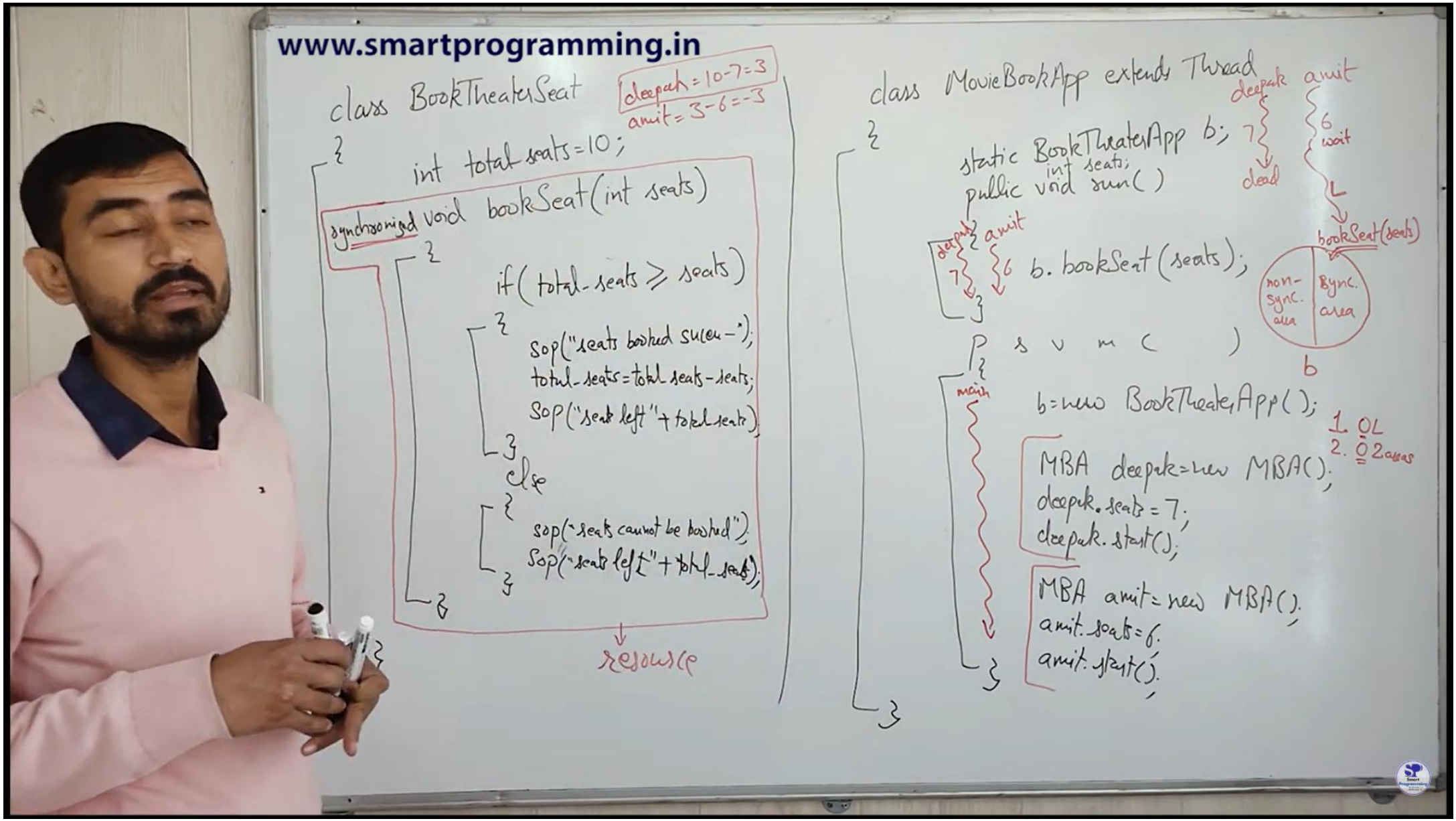
1. By "Synchronized Method"
2. By "Synchronized Block"
3. By "Static Synchronization"

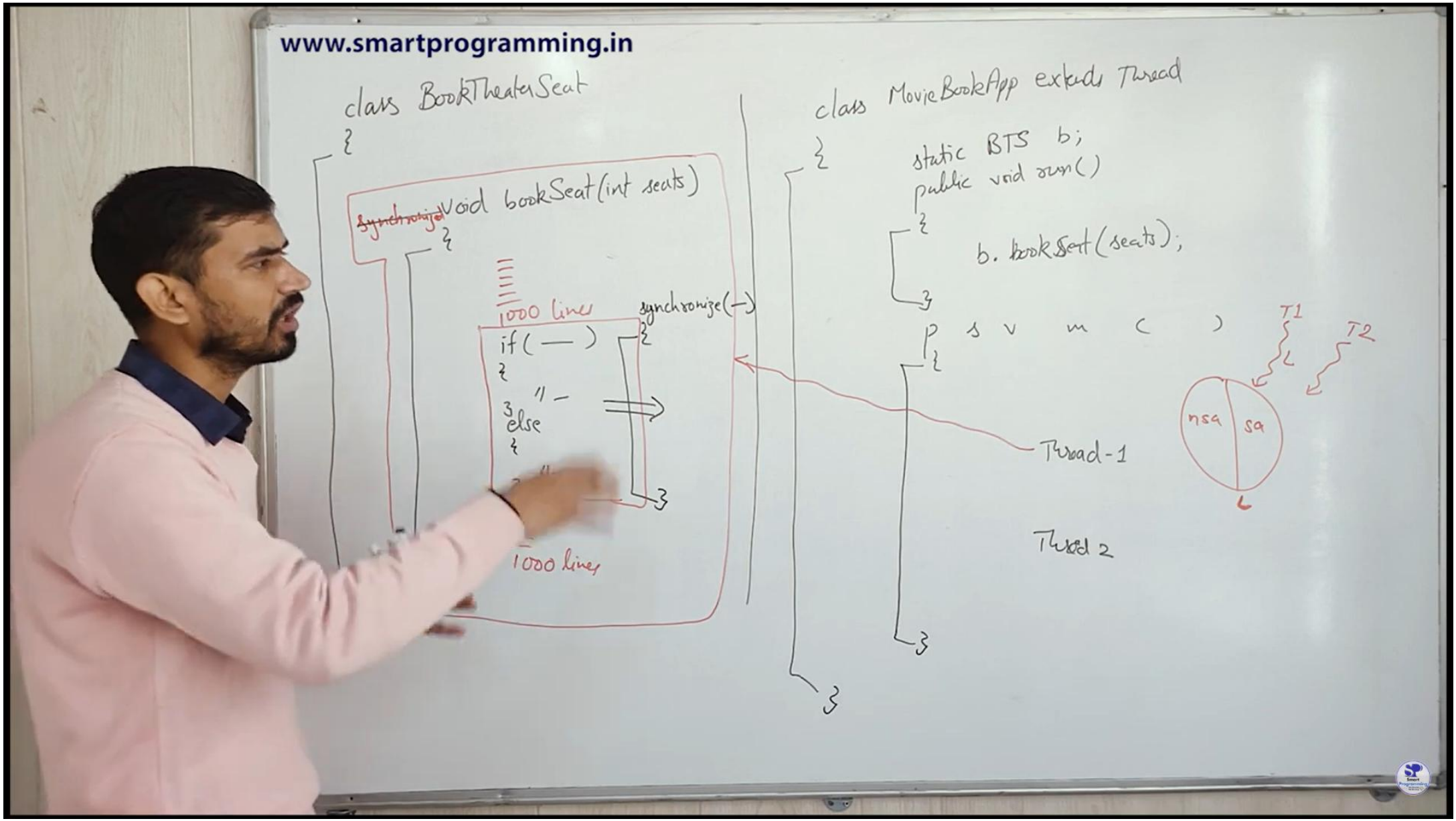
Cooperation

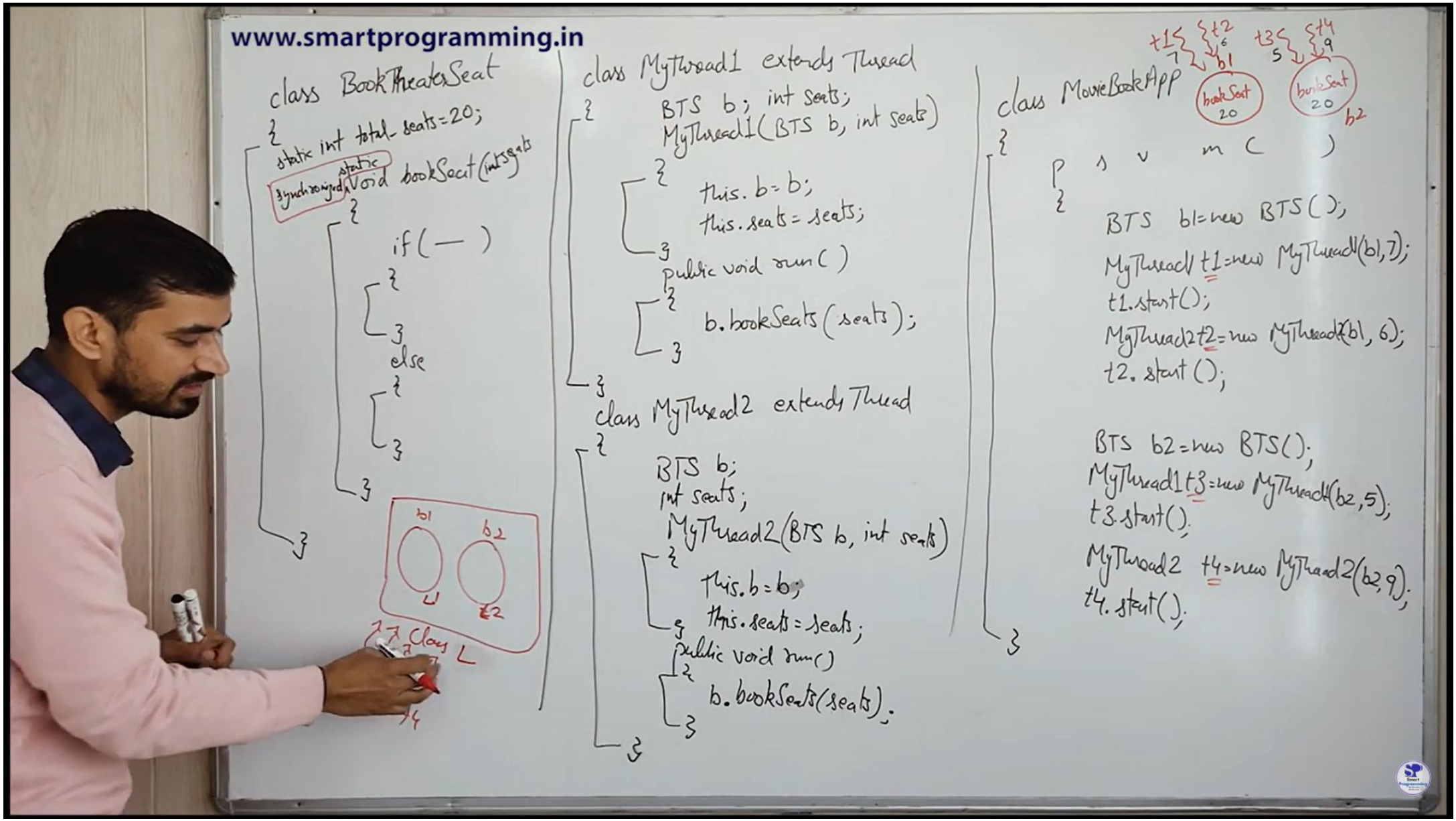
(Inter-thread communication in java)

Can be achieved by following methods of Object class:

1. wait()
2. notify()
3. notifyAll()


www.smartprogramming.in
class BookTheaterSeat
deepak = 10-7=3
amit = 3-6=-3
{
int total_seats=10;
synchronized void bookSeat(int seats)
{
if(total_seats >= seats)
{
Sop("seats booked succes-");
total_seats=total_seats-seats;
Sop("seat left"+total_seats);
}
else
{
sop("seats cannot be booked");
Sop("seats left"+total_seats);
}
}
}
resource
class MovieBookApp extends Thread
{
static BookTheaterApp b;
int seats;
public void run()
{
b.bookSeat(seats);
}
P s v m ( )
{
main
b=new BookTheaterApp();
MBA deepak=new MBA();
deepak.seats=7;
deepak.start();
MBA amit=new MBA();
amit.seats=6;
amit.start();
}
}
deepak
amit
7
dead
6
wait
bookSeat(seats)
non-sync. area
Sync. area
b
1. OL
2. O 2 areas


www.smartprogramming.in
class BookTheaterSeat
{
synchronized void bookSeat(int seats)
{
1000 lines
synchronize(—)
if(—)
{
}
else
{
}
1000 lines
class MovieBookApp extends Thread
{
static BTS b;
public void run()
{
b.bookSeat(seats);
}
p s v m ( )
{
}
Thread-1
Thread 2
T1
T2
nsa
sa


www.smartprogramming.in
class BookTheatreSeat
{
static int total_seats=20;
synchronized static void bookSeat(int seats)
{
if(—)
{
}
else
{
}
}
}
b1
b2
t1
t2
class MyThread1 extends Thread
{
BTS b; int seats;
MyThread1(BTS b, int seats)
{
this.b=b;
this.seats=seats;
}
public void run()
{
b.bookSeats(seats);
}
}
class MyThread2 extends Thread
{
BTS b;
int seats;
MyThread2(BTS b, int seats)
{
this.b=b;
this.seats=seats;
}
public void run()
{
b.bookSeats(seats);
}
}
class MovieBookApp
{
p s v m ( )
{
BTS b1=new BTS();
MyThread1 t1=new MyThread1(b1,7);
t1.start();
MyThread2 t2=new MyThread2(b1,6);
t2.start();
BTS b2=new BTS();
MyThread1 t3=new MyThread1(b2,5);
t3.start();
MyThread2 t4=new MyThread2(b2,9);
t4.start();
}
}
t1
7
t2
6
b1
bookSeat
20
t3
5
t4
9
bookSeat
20
b2

Inter-thread Communication

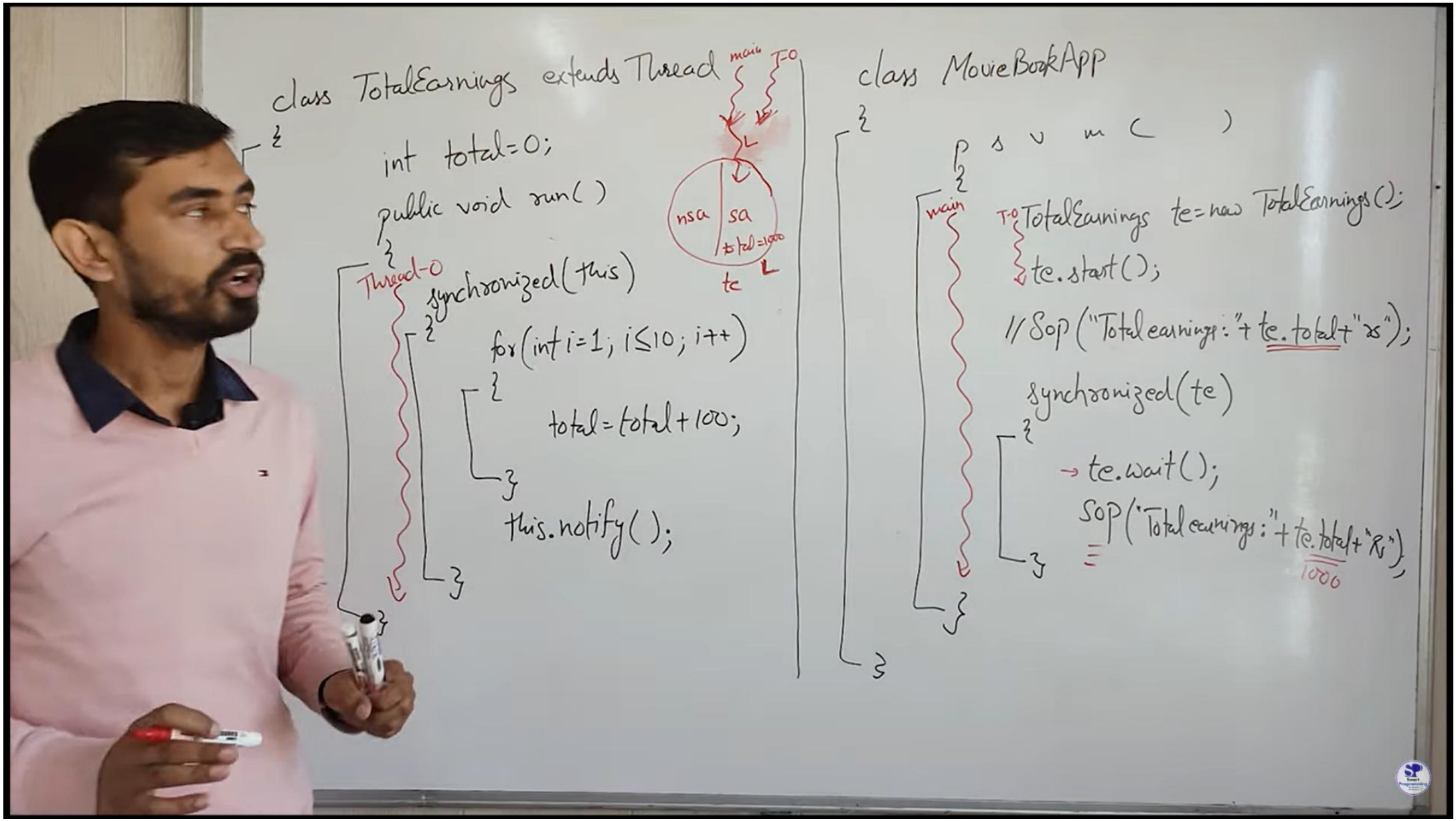
Inter-thread communication is a mechanism in which a thread releases the lock and enter into paused state and another thread acquires the lock and continue to executed.
It is implemented by following methods of Object class:

1. wait() :- If any thread calls the wait() method, it causes the current thread to release the lock and wait until another thread invokes the notify() or notifyAll() method for this object, or a specified amount of time has elapsed.
Syntax : public final void wait()throws InterruptedException (waits until object is notified)
public final void wait(long timeout)throws InterruptedException (waits for the specified amount of time)

2. notify() : This method is used to wake up a single thread and releases the object lock
Syntax : public final void notify()

3. notifyAll () : This method is used to wake up all threads that are in waiting state.
Syntax: public final void notifyAll()

NOTE : To call wait(), notify() or notifyAll() method on any object, thread should own the lock of that object i.e. the thread should be inside synchronized area


class TotalEarnings extends Thread
int total=0;
public void run()
synchronized(this)
for(int i=1; i≤10; i++)
total=total+100;
this.notify();
Thread-0
main
T-0
nsa
sa
total=1000
te
class MovieBookApp
p s v m ( )
TotalEarnings te=new TotalEarnings();
te.start();
//Sop("Total earnings:"+te.total+"/-");
synchronized(te)
te.wait();
Sop("Total earnings:"+te.total+"/-");
1000