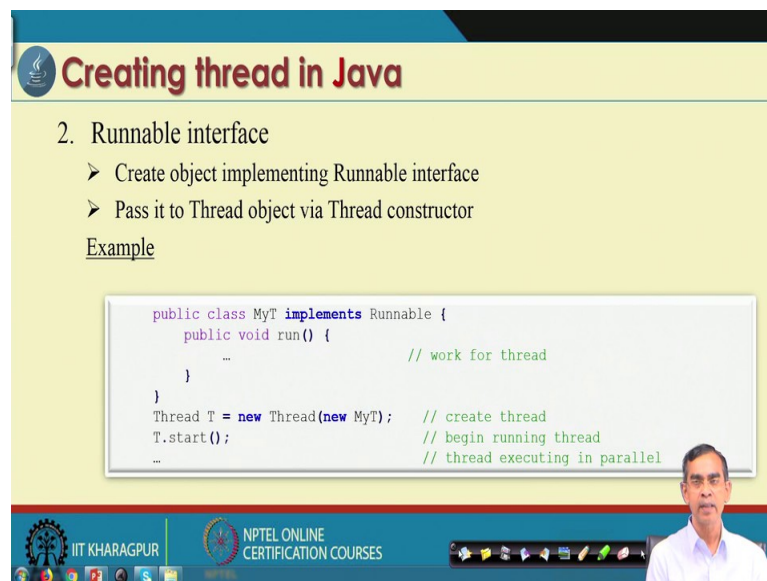


**Programming in Java**  
**Prof. Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 28**  
**Multithreading – II**

So, in the last module related to multithreaded Programming in Java, we have discussed about the basic concept of Multithreading and then how the thread can be created in Java program and then how the different threads can be executed in parallel. And in the last discussion, we have used the class thread to create our thread. Now in this discussion, we will discuss about how the Thread can be created using the Runnable interface the procedure is very similar to the Thread class creation only a few minor difference is there.

(Refer Slide Time: 00:53)



**Creating thread in Java**

2. Runnable interface

- Create object implementing Runnable interface
- Pass it to Thread object via Thread constructor

Example

```
public class MyT implements Runnable {  
    public void run() {  
        ... // work for thread  
    }  
}  
Thread T = new Thread(new MyT); // create thread  
T.start(); // begin running thread  
... // thread executing in parallel
```

The slide includes a footer with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video feed of the professor in the bottom right corner.

Now, let us first start with how we can create a thread using Runnable interface. As you know if we want to create our Thread using class Thread so, we have to create a Thread class which is basically extending the class thread. On the other hand the same procedure, but here we have to create a Thread class that implements Runnable. Now implements Runnable means in the Runnable interface the method run method is there which is the public and abstract method these method needs to be created. So, it is

basically implementation of this run Thread is similar to the implementation of the run method that is there in the Thread class here.

So, in that case, the Thread class run method they are in if you have to extend Thread class is overriding, but here the run method is to be implemented. Once this class Thread class is created by Runnable interface then we can create a Thread object for example, here T is the Thread objects and when you create the Thread objects you see we use the Thread class and pass these as a parameter. So, my Thread that you have created a new my two all object is needed to be a pass. So, this is the only difference that you have to these.

In other cases, if you use the Thread class so, Thread to new Thread no argument to be passed. So, the only default constructor will take place, but here in this if you use a Runnable or implement the Thread class using Runnable interface then that object needs to be passed here, that is the only difference otherwise everything is same basically. So, here T dot start is basically to start the execution of this thread. Now it is basically the same again that this is the run method that we have to define as it is we have defined there in the creating class using extension of the Thread class and then T start.

So, both the things are, here again, now let us again repeat the semi same procedure that we have discussed about in ok. In our last module exam example, we have considered about 3 threads Thread a, Thread b, Thread c, which basically p negative number, even number and odd number. The same concept here also we are we will just implement each, but using Runnable interface so, that we can understand that how the two things are work for us.

(Refer Slide Time: 03:22)


### Creating thread : Example

```
class ThreadX implements Runnable
{
    public void run() {
        for(int i = 1; i <= 5; i++) {
            System.out.println("Thread X with i = "+ i*i);
        }
        System.out.println("Exiting Thread X ...");
    }
}

class ThreadY implements Runnable {
    public void run() {
        for(int j = 1; j <= 5; j++) {
            System.out.println("Thread Y with j = "+ 2*j);
        }
        System.out.println("Exiting Thread Y ...");
    }
}
```

```
class ThreadZ implements Runnable
{
    public void run() {
        for(int k = 1; k <= 5; k++) {
            System.out.println("Thread Z with k = "+ 2*k-1);
        }
        System.out.println("Exiting Thread Z ...");
    }
}

class MultiThreadRunnable {
    public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);
        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);
        Thread t3 = new Thread(new ThreadZ());
        t1.start();
        t2.start();
        t3.start();
        System.out.println("... Multithreading is over");
    }
}
```



So, here exactly let see this is the Thread X I have given the name Thread X, in this case, implements Runnable. So, this is the similar the run code that we have used earlier and then Thread Y same as Thread b that we have discussed here, but it implements the Runnable run method here.

(Refer Slide Time: 03:48)

### Creating thread : Example

```
class ThreadX implements Runnable
{
    public void run() {
        for(int i = 1; i <= 5; i++) {
            System.out.println("Thread X with i = "+ i*i);
        }
        System.out.println("Exiting Thread X ...");
    }
}

class ThreadY implements Runnable {
    public void run() {
        for(int j = 1; j <= 5; j++) {
            System.out.println("Thread Y with j = "+ 2*j);
        }
        System.out.println("Exiting Thread Y ...");
    }
}
```

```
class ThreadZ implements Runnable
{
    public void run() {
        for(int k = 1; k <= 5; k++) {
            System.out.println("Thread Z with k = "+ 2*k-1);
        }
        System.out.println("Exiting Thread Z ...");
    }
}

class MultiThreadRunnable {
    public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);
        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);
        Thread t3 = new Thread(new ThreadZ());
        t1.start();
        t2.start();
        t3.start();
        System.out.println("... Multithreading is over");
    }
}
```

So, ThreadX. ThreadY are declared and then it will implement the Thread Z. So, this is the Thread Z. So, 3 Thread ThreadX ThreadY and ThreadZs are implemented. So, created we can say 3 Thread class are created by implementing Runnable interface and

this is the main method as we see here these are the main class and here you can see we create x y and x and y are the Thread object of this Thread x class. And then Thread is basically created t 1 bypassing this x because, if we use a Runnable interface that object needs to be passed as a constructor to call the I mean Thread class actually this is a constructor that is defined in the Thread class.

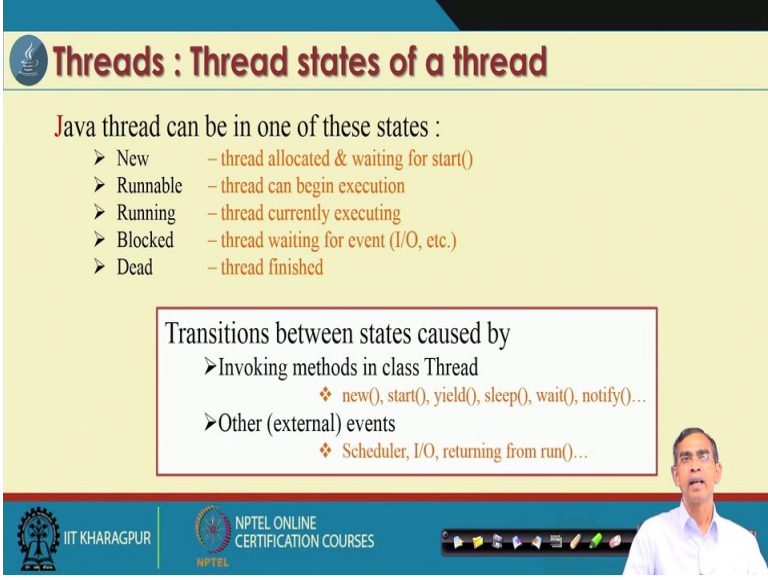
So, these ways we can see t 1 t 2 and t 3 as you can see in this case again we can call in one here actually two-step process but is a single-step process that means we can create the Thread directly call this these passing these also it is equally applicable here. So, what we can see here t 1 t 2 and t 3 3 Threads objects are created and one these 3 Thread objects are created they are ready for execution and these are the basically to start the Thread we use t 1 dot start t 2 dot start t 3 dot start and then Thread will be run in parallel. So, this is the multithreading using Runnable interface as we have learned it here.

So, we can see basically the same thing implements versus extends and then just how to create the Thread object otherwise for the run and everything the same in both concepts. So, the two ways of running a multithreading program using extends and Runnable interface.

So, there is no I mean such a rule that these were these way the search basically the difference syntax we can say the different procedure how the different Thread can executes now. So, we have learned about how a Thread can be created although it is a simple example that we have considered, this is the idea about and then it can be extended to any type of complex there any number of threads can be executed any number of threads can be then started to execute.

Now there are many more things are there particularly a Thread can be in a different state. So, now, we will discuss about what are the different states that a Thread may have and this is very important because whenever we have to control a very complex program you should know the different states of a Thread that it may have and then how we can communicate one Thread to another.

(Refer Slide Time: 06:33)



**Threads : Thread states of a thread**

Java thread can be in one of these states :

- New – thread allocated & waiting for start()
- Runnable – thread can begin execution
- Running – thread currently executing
- Blocked – thread waiting for event (I/O, etc.)
- Dead – thread finished

Transitions between states caused by

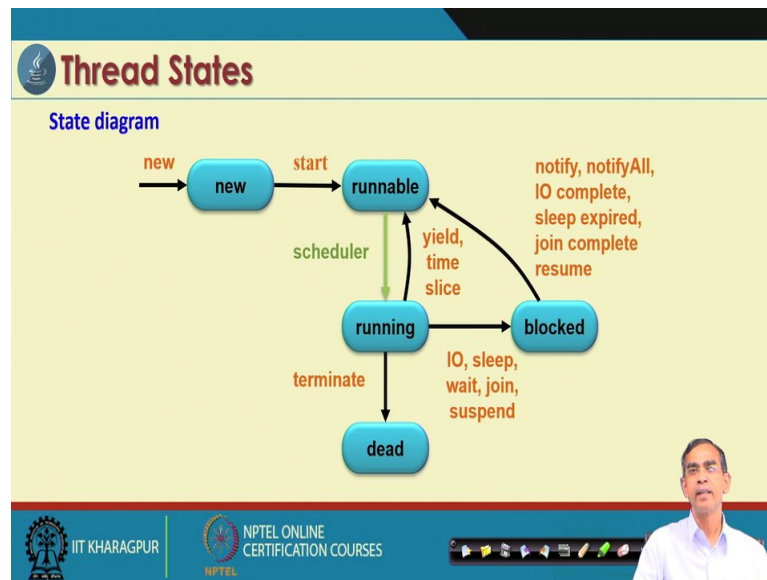
- Invoking methods in class Thread
  - ❖ new(), start(), yield(), sleep(), wait(), notify()...
- Other (external) events
  - ❖ Scheduler, I/O, returning from run()...

The slide also features the IIT Kharagpur and NPTEL logos at the bottom left, a navigation bar at the bottom center, and a small video inset of a speaker at the bottom right.

So, basically the interfaces inter-process communication. Now, so, far a Thread is concerned it has different states like Thread can be in a new state. So, this mean the Thread is just created and it is yet to start its execution. So, it is a new Runnable. So, Thread is basically being execution that means it is about to run and run this is basically active the Thread is in the execution and blocked somehow Thread need something or Thread has been stopped or it is slipping so; that means, Thread is in a waiting state it is called the blocked state and then finally, a Thread can be in a dead state; that means, when it finished it execution.

So, these are the 5 different states that a Thread can be in this there and there are different methods by which a Thread can change its state like a new, start, yield, sleep, wait and notify and other some external events also their scheduler, I O and returning from run and resume all these things are there.

(Refer Slide Time: 07:51)



Now, let us have a detailed discussion detail account of all these methods which basically to control the states of a threads are there, now this is a state transition diagram for a Thread that it may have as we have checked that a Thread can be in a new state. So, whenever the created and then this Thread can be in a new state this means Thread is not yet started its execution, but ready to start its execution. Now when the start method is called then this Thread is basically Runnable but is a Runnable basically gives to the scheduler it is basically the Java run time manager it basically all the Threads are now started it but is about to execute or run it.

When the scheduler can call it is basically yield and then time slice is allowed to each state may be in a time-sharing manner that it will share the different resources CPU all these things so on. So, this is totally is a paradigm of Java run time manager which we should not bother about anything, but it is basically working like this one. So, whenever it is there so, it is running. So, there again once the Thread is running that Thread can be stopped or can be resumed or it can be terminated or suspense so, it is basically terminated.

So, once that terminated the Thread will go to the dead state otherwise if the Thread is waiting for any resources in that it required may be Thread is waiting for some other disk or memory that it is waiting. So, these are the running to it can go to the waiting state. So, blocked in a running to waiting state the different methods it is there I O request

sleep, wait, join, suspend these are the methods which can change the state from running to block state. From the block state it can again Runnable different methods, which are responsible to move a state from blocked state to Runnable states are here.

So, these are the way that the Thread can change from new to dead state and the different processes are there. So, this is basically state-run transition diagram says that how the state of a Thread can change and what are the communicating methods are there to control the execution of a threads and then Thread can be in a different state.

(Refer Slide Time: 10:02)

### Thread control methods

- `start()` :→ A newborn thread with this method enter into **Runnable** state and Java run time create a system thread context and starts it running. **This method for a thread object can be called once only**
- `suspend()` :→ This method is different from **stop()** method. It takes the thread and causes it to stop running and later on can be restored (by **resume()**)
- `resume()` :→ This method is used to revive a suspended thread. There is no gurantee that the thread will start running right way, since there might be a higher priority thread running already, but, **resume()** causes the thread to become eligible for running
- `sleep(int n)`:→ This method causes the run time to put the current thread to sleep for **n milliseconds**
- `yield()` :→ This method causes the run time to switch the context from the current thread to the next available runnable thread. This is one way to **ensure that the threads at lower priority do not get started**

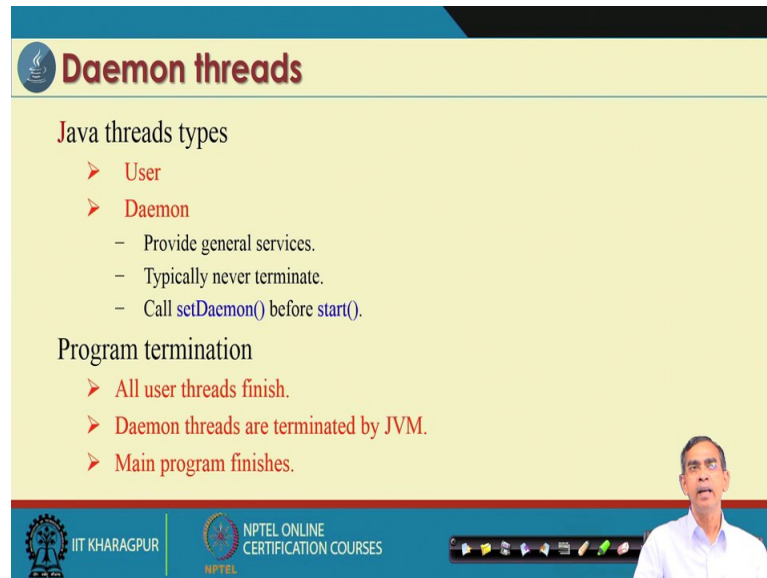
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, these are the different methods that we have learned about the start method we have already used it and then suspend method. Suspend method is not exactly the stopping stop method means the Thread will be dead but suspend means it will temporarily halt the execution and then if a Thread is suspended by the calling suspend method it can be again revoked using resume method.

Now, the resume method is basically when the Thread is in a waiting state or in a blocked state it can be turned into Runnable state. So, the schedule scheduler can take all the threads from the Runnable state to assign to the CPU for execution and then the sleep method is basically by explicitly maintaining that Thread should stop it here wait for some seconds may be n milliseconds like where the argument is n there. So, sleep means the Thread will not do anything only the sleep for sometimes and yield method is basically switching that takes from the context of a Thread from the current Thread to the

next available running Thread that is their Runnable thread. So, it is basically usually this yield method used by the scheduler to invoke a Thread which is waiting ready for execution that is there in the Runnable state.

(Refer Slide Time: 11:25)



**Daemon threads**

Java threads types

- User
- Daemon
  - Provide general services.
  - Typically never terminate.
  - Call `setDaemon()` before `start()`.

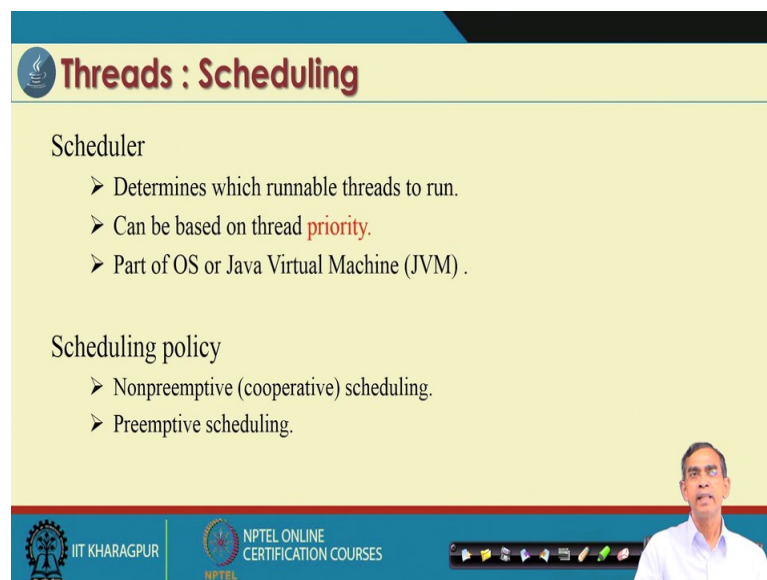
Program termination

- All user threads finish.
- Daemon threads are terminated by JVM.
- Main program finishes.

The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom, and a small video inset of a speaker in the bottom right corner.

Now, let us see how the thread's scheduling can be done.

(Refer Slide Time: 11:27)



**Threads : Scheduling**

Scheduler

- Determines which runnable threads to run.
- Can be based on thread **priority**.
- Part of OS or Java Virtual Machine (JVM) .

Scheduling policy

- Nonpreemptive (cooperative) scheduling.
- Preemptive scheduling.

The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom, and a small video inset of a speaker in the bottom right corner.

And Thread scheduling can be done there are actually scheduler through scheduler is basically responsible for scheduling a Threads; that means, scheduling a Thread means it

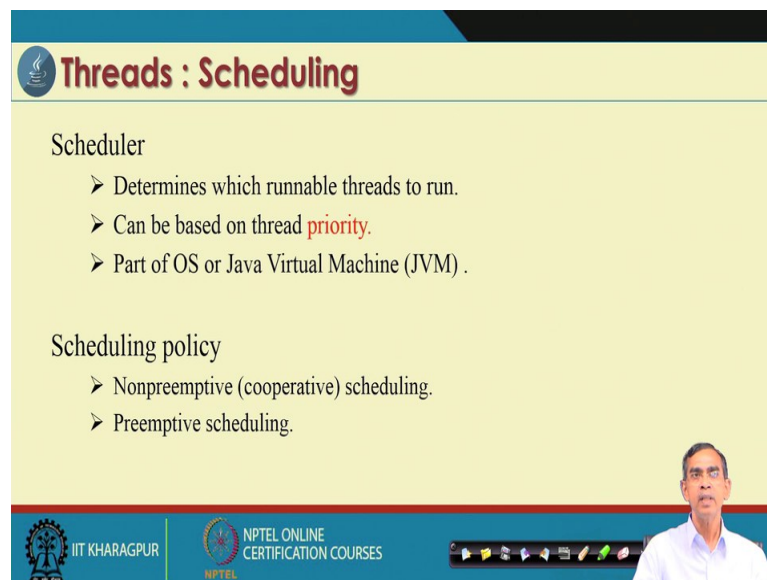


basically assign Threads to Java run time manager and is basically determines that which Threads which are there in a Runnable state can be assigned to the system for execution.

Now, again in the Java system, we can assign the priority low, mean priority, high priority, max priority like. So, according to the priority scheduler will call revoke a method Thread from the Runnable state to this one and there are some threads we call the demo thread. So, demo threads are basically running all the time in the system and then the scheduler can interact with them so, that they can be executed and then they can control their execution and they the Thread can be in any other states there.

So, the demo Daemon Thread is basically for the general service whenever suppose you just insert one disk. So, automatically it detect that disk is there whenever some interrupts are coming to the daemon Thread will take these are the few examples are there. Now the program can be terminated whenever all threads are finished its execution if there is some Thread is yet to be executed the program cannot be terminated and then we have to do for the force termination by pressing the power button or like this one.

(Refer Slide Time: 12:58)



**Threads : Scheduling**

**Scheduler**

- Determines which runnable threads to run.
- Can be based on thread **priority**.
- Part of OS or Java Virtual Machine (JVM) .

**Scheduling policy**

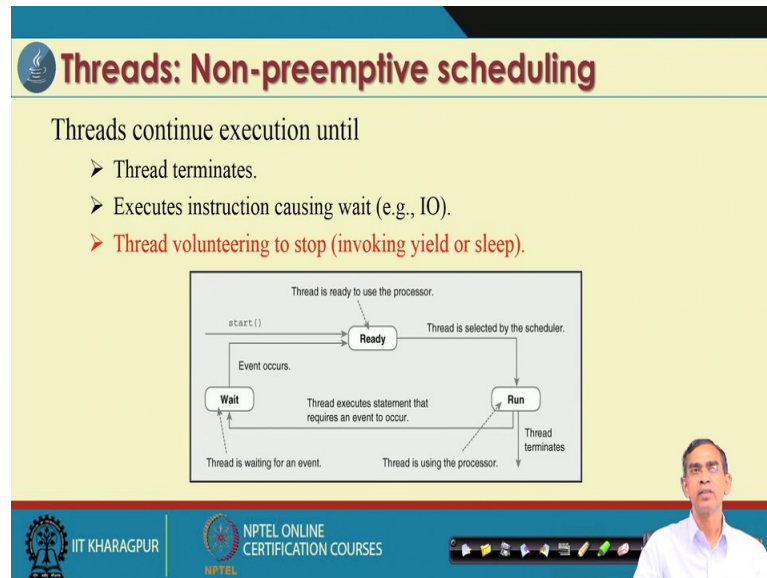
- Nonpreemptive (cooperative) scheduling.
- Preemptive scheduling.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Anyway so, whenever the scheduling of a Thread is concerned again in the system there are two type of scheduling called the pre-emptive scheduling and non pre-emptive, non pre-emptive means one the Thread is in execution if it is a non pre-emptive scheduling it is followed then this Thread cannot be interrupted until it finish it will start its execution. So, no it cannot be interrupted whereas, the pre-emptive scheduling means, Thread is in

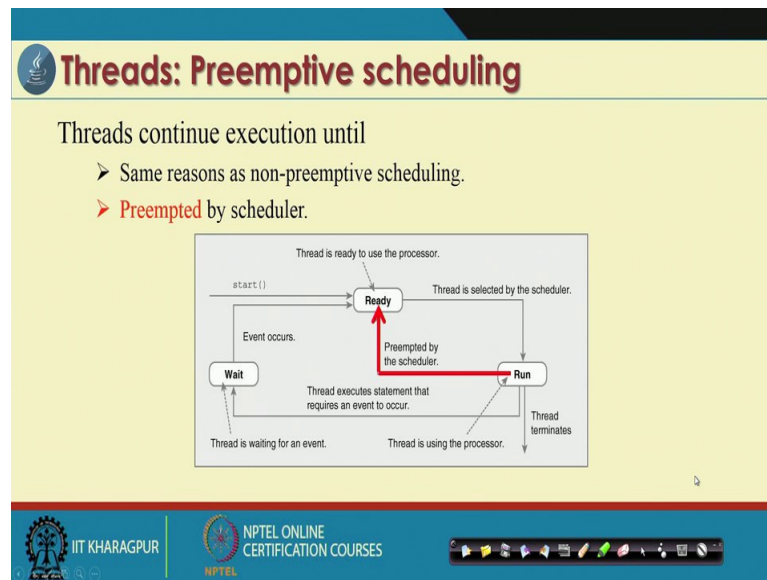
execution, it can be pre-empted may be either can change its state from running to waiting for state or sleep or whatever it is there.

(Refer Slide Time: 13:38)



So, it is basically the policy of how you want to control the threads in your execution. So, this is an idea about the pre-emptive scheduling that it basically. So, whenever the Thread is in running condition, it will either terminates or it will go wait or ready wait or ready until it finishes its execution. So, this is the idea about the pre non pre-emptive scheduling; that means, it will no interaction interruption is allowed in case of non pre-emptive scheduling is there on the other hand in case of pre-emptive scheduling as we see.

(Refer Slide Time: 14:23)



As we see the pre-emptive scheduling is basically if there is an interruption occur. So, it basically not wait it will from the run state to go to the ready state. So, this way pre-emptive occurs anyway. So, this is the matter of policy that you can implement in your system and this is the concept of an idea about managing the systems whenever you have to implement it. So, pre-emptive non pre-emptive the core concept of how you want to control your process in execution or we can say the Thread in execution.

(Refer Slide Time: 14:56)

### Java thread : An example

```
public class ThreadExample extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            try {
                sleep ((int) (Math.random() * 5000)); // 5 secs
            }
            catch (InterruptedException e) {
                System.out.println (i);
            }
        }
    }
    public static void main(String[] args) {
        new ThreadExample().start();
        new ThreadExample().start();
        System.out.println ("Done");
    }
}
```

Now, let us have an example of how the different methods are there which can be implied to a Thread whenever it is in execution, this is again a simple method that we can understand about it and it illustrates how the sleep method will work for you. So, here is an example we can see we create a Thread class the name of the Thread class is ThreadExample which basically created using Thread class. So, extends Thread now I can tell you using implement Runnable also you can do it anyway.

So, here is the run method as you see how we can code the run method then this one it is basically is a loop for loop and here you see sleep we use the sleep method for this that mean whenever it executed this for i equal to 0 it will just call the sleep method and it will sleep for 5000 millisecond. So, 5 seconds like and we put this code into this one because here there may be an exception occurs that 3 Thread is no more there and then we want to call this Thread using sleep method. So, this is why it is better to put under try-catch. So, it is basically interrupted exception if it occurs then it can be caught it and then handle from the premature termination of the program itself or the robust program this is basically exception handling concept it is followed there.

Now, as we see. So, this is the run method this run method is just run the Thread and here the idea is that Thread will for each loop whenever it runs. So, first 5seconds it will sleep then again 5 second it will sleep and then another 5 seconds it will be sleeping like this one. So, it is the idea about sleep there and if you print one system statement `system.out.print l n` before here and after here that before sleeping first and then after sleeping first then you can see that. So, the threads are executed in parallel wherever it is sleeping intermittent way.

So, here is basically the idea about here we can see we have created two Thread ThreadExample object and then start there so; that means, we run two threads parallely and if we run this threads parallely with their print statement here in our demonstration we will give a demo how this Thread can execute in a different order and everything. So, this will give an idea about that the threads are running and invoking the sleep method; that means, Thread is going to sleep there. So, basically, here the threat Thread that we have created here we are sending a method to control or just change its state from may be running to waiting for a state like this one or blocking state whatever it is.

So, this way the sleep method like this sleep method we can use many other methods as we have discussed in previous slides namely, suspend all these things stop all this things also we can call it here. So, appropriate code is to be planned for that to have it any way we will discuss all these things in our demo. So, that how the different methods can be called and then how their consequence is there.

(Refer Slide Time: 18:20)

**Java Thread Example**

Possible outputs

- 0,1,2,0,1,2,Done // thread 1, thread 2, main()
- 0,1,2,Done,0,1,2 // thread 1, main(), thread 2
- Done,0,1,2,0,1,2 // main(), thread 1, thread 2
- 0,0,1,1,2,Done,2 // main() & threads interleaved

Diagram illustrating thread execution flow:

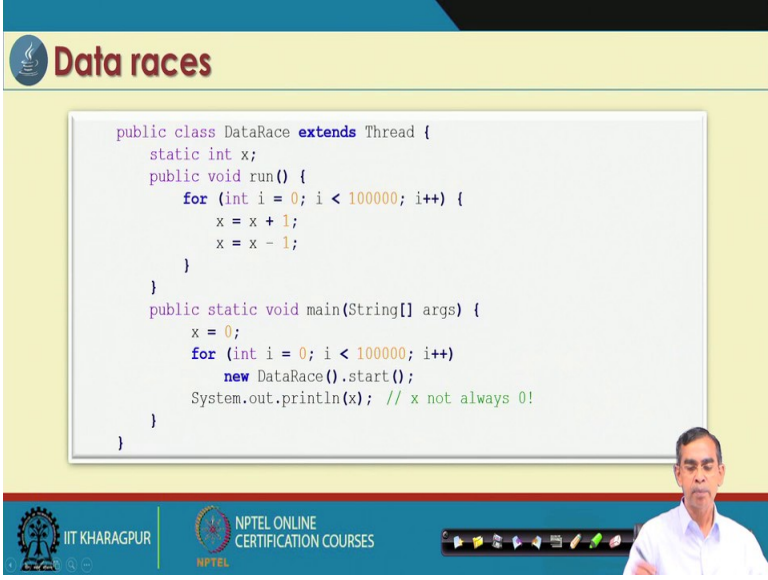
```
graph TD
    Main["main(): thread 1, thread 2, println Done"]
    Thread1["thread 1: println 0, println 1, println 2"]
    Thread2["thread 2: println 0, println 1, println 2"]
    Main --- Thread1
    Main --- Thread2
    Thread1 --- Thread2
```

The diagram shows three boxes representing the execution of the main method, thread 1, and thread 2. The main method box is at the top, with arrows pointing down to the thread 1 and thread 2 boxes. The thread 1 box is to the left of the thread 2 box, and both have arrows pointing to each other, indicating interleaved execution.

Footer: IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so, this is an example again we will discuss the parallel execution of this statement how this state can be executed when the sleep condition I want to skip here. So, for the theoretical it is more aspect. So, far that how it basically runs, it in actual programming environment ok. So, this is a Thread control we can say and these are basically the matter the concept or the features.

(Refer Slide Time: 18:49)



### Data races

```
public class DataRace extends Thread {
    static int x;
    public void run() {
        for (int i = 0; i < 100000; i++) {
            x = x + 1;
            x = x - 1;
        }
    }
    public static void main(String[] args) {
        x = 0;
        for (int i = 0; i < 100000; i++)
            new DataRace().start();
        System.out.println(x); // x not always 0!
    }
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

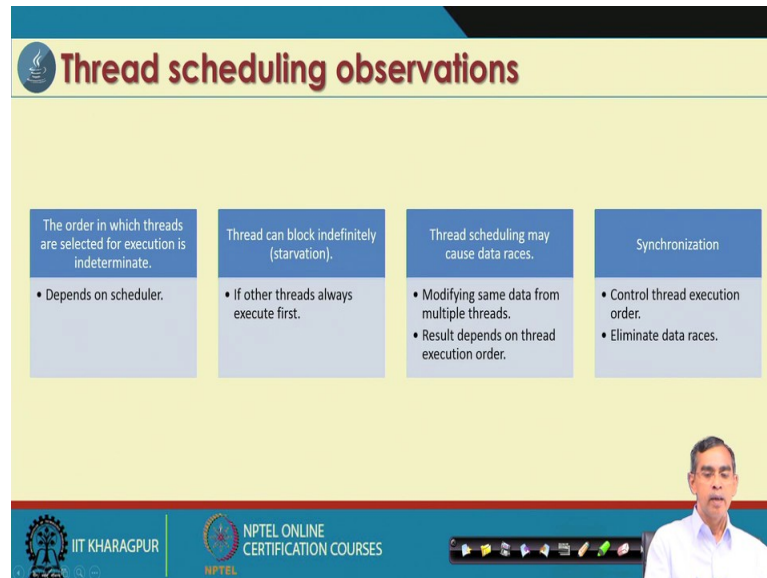
So, for the inter-process Thread communication is concerns here and this is another example of again inter-process communication of the Thread as we see and this is a very interesting problem which usually occurs whenever multiple threads are in the execution of the concept is called the DataRaces. So, here is basically we create one class called the DataRace which extends the Thread class here and here we declare one method run which uses one static integer variable x there. Now, this method is basically used interchanging for integer i equal to 0, i less than these are basically. So, many times this will loop and for each iteration x equal to x plus 1 and x equal to x minus 1 as if this loop is executed.

So, for every iteration x will be initially if x is 0. So, it is 1 then 0 1 and 0, but interestingly if you run the Thread this DataRace Thread and you see it it should in this according to this one single Thread it will always keep the 0 as a value of x, but here the thing is that because of this intermittent execution and then what is called data races are there. So, if we run this Threads for this many loops and you will see it always not give 0 it sometimes keeps 1 sometimes gives 0, this is because of the different Thread whenever it is execution and it is a static variable.

So, it prints the different situation whatever the values are there in this intermittent execution. So, this is the idea about and then and it basically calls the data race example

in the Java programming concept. So, this basically says that how the different threads are in execution that we can check it within this kind of illustration.

(Refer Slide Time: 20:47)



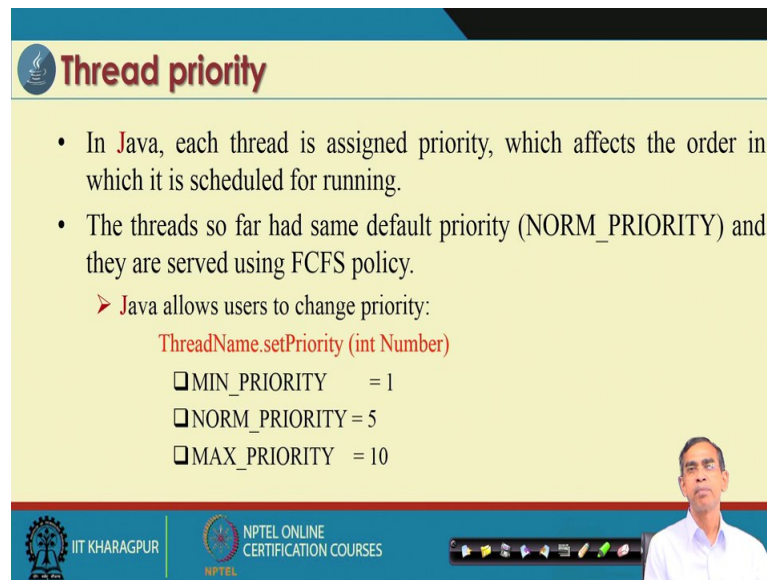
**Thread scheduling observations**

- The order in which threads are selected for execution is indeterminate.
  - Depends on scheduler.
- Thread can block indefinitely (starvation).
  - If other threads always execute first.
- Thread scheduling may cause data races.
  - Modifying same data from multiple threads.
  - Result depends on thread execution order.
- Synchronization
  - Control thread execution order.
  - Eliminate data races.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so, these are the different way the Thread can be scheduled our Thread can be communicated inter-process communication we can say and now next is that a Thread can be synchronized also. So, the Thread synchronization and before this Thread synchronization, there is also one concept is called the setting the priority our next example is basically how the set priority of a thread can be established so, that the scheduler can schedule the execution or the Thread according to their priority.

(Refer Slide Time: 21:23).



**Thread priority**

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running.
- The threads so far had same default priority (NORM\_PRIORITY) and they are served using FCFS policy.
- Java allows users to change priority:  
`ThreadName.setPriority (int Number)`
  - ☐ MIN\_PRIORITY = 1
  - ☐ NORM\_PRIORITY = 5
  - ☐ MAX\_PRIORITY = 10

The slide features a blue header with the title 'Thread priority' in red. The main content is on a yellow background. At the bottom, there is a blue footer with logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL, along with a small video inset of a man in a white shirt.

Now, so far the priority of a Thread is concerned there are three different priorities that can be there in the Thread MIN PRIORITY, NORM PRIORITY and then MAX PRIORITY. So, they have their global value called 1 5 and 10. So, we can set the priority of a Thread by calling this method set priority which is defined therein class Thread. So, if this is the Thread and then we can call this method setPriority and passing the num number 1 5 or 10, then it will set the priority according to MIN PRIORITY NORM PRIORITY or MAX PRIORITY, otherwise you can write this one min priority the global variable declared in the Thread class itself. So, we can use it also.

So, a Thread can be assigned its priority and once it is assigned then the Thread can be served using some CPU scheduling policy, for example, FCFS First Come First Served policy look like this one. So, there is the way that the Thread can be there now let us have an example.






























(Refer Slide Time: 22:32)

### Thread priority : An example

```
class A extends Thread
{
    public void run()
    {
        System.out.println ("Thread A started");
        for (int i=1;i<=4;i++)
        {
            System.out.println ("\t From ThreadA: i= "+i);
        }
        System.out.println ("Exit from A");
    }
}
```

```
class B extends Thread
{
    public void run()
    {
        System.out.println ("Thread B started");
        for (int i=1;i<=4;i++)
        {
            System.out.println ("\t From ThreadB: i= "+i);
        }
        System.out.println ("Exit from B");
    }
}
```

```
class C extends Thread
{
    public void run()
    {
        System.out.println ("Thread C started");
        for (int i=1;i<=4;i++)
        {
            System.out.println ("\t From ThreadC: i= "+i);
        }
        System.out.println ("Exit from C");
    }
}
```



So, that how this Thread can be priority can be assigned and accordingly the Thread execution can be controlled. So, here is an example of this. So, here class A we create a Thread and then run method we define here run method has a loop and this is a loop control variable i equals to 1 to 4. So, loop will roll for times and each time the loop is rolled it will print the value of i here; that means, loop variables. So, this is the Thread A and similarly, this is the class A basically one Thread that we have created, now let us have some more Thread because multiple Thread execution is more interesting.




























(Refer Slide Time: 23:18)

### Thread priority : An example

```
class A extends Thread
{
    public void run()
    {
        System.out.println ("Thread A started");
        for (int i=1;i<=4;i++)
        {
            System.out.println ("\t From ThreadA: i= "+i);
        }
        System.out.println ("Exit from A");
    }
}
```

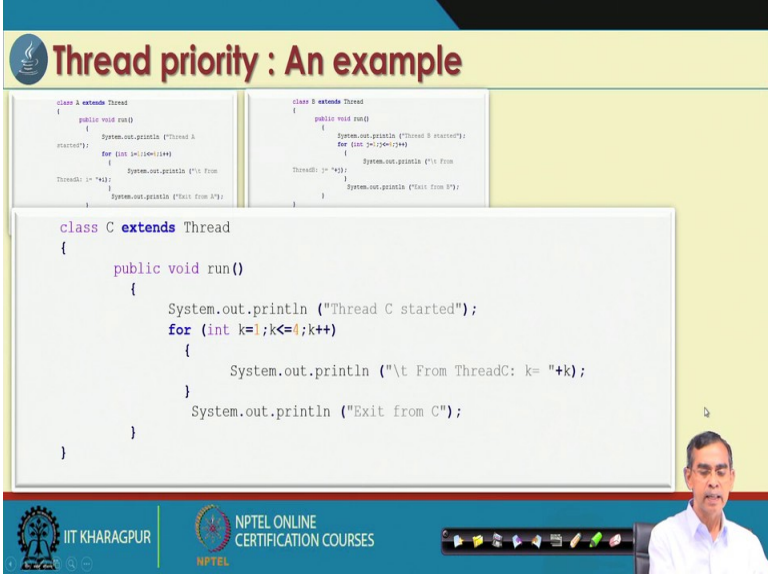
```
class B extends Thread
{
    public void run()
    {
        System.out.println ("Thread B started");
        for (int j=1;j<=4;j++)
        {
            System.out.println ("\t From ThreadB: j= "+j);
        }
        System.out.println ("Exit from B");
    }
}
```

```
class C extends Thread
{
    public void run()
    {
        System.out.println ("Thread C started");
        for (int i=1;i<=4;i++)
        {
            System.out.println ("\t From ThreadC: i= "+i);
        }
        System.out.println ("Exit from C");
    }
}
```



So, here is another example and here is a class B another Thread that you have created this also similar to the class A Thread, but it just printing the variable the loop variable of its own. And so, this is the class B and similarly, we can define another class say class C say another Thread.

(Refer Slide Time: 23:38)



### Thread priority : An example

```
class A extends Thread
{
    public void run()
    {
        System.out.println ("Thread A started");
        for (int i=1;i<=10;i++)
        {
            System.out.println ("i From ThreadA: "+i);
        }
        System.out.println ("Exit from A");
    }
}

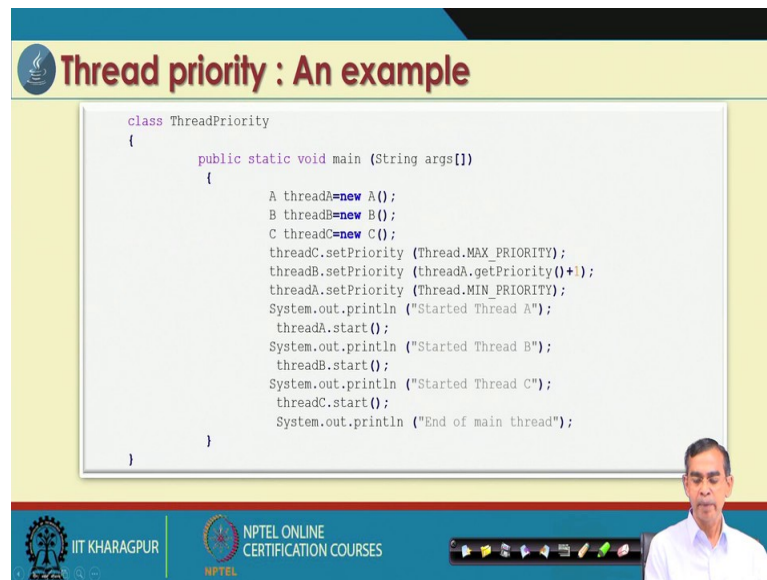
class B extends Thread
{
    public void run()
    {
        System.out.println ("Thread B started");
        for (int j=1;j<=10;j++)
        {
            System.out.println ("j From ThreadB: "+j);
        }
        System.out.println ("Exit from B");
    }
}

class C extends Thread
{
    public void run()
    {
        System.out.println ("Thread C started");
        for (int k=1;k<=10;k++)
        {
            System.out.println ("\t From ThreadC: k= "+k);
        }
        System.out.println ("Exit from C");
    }
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And this is a class C Thread as we see class C Thread and then this also another a run method it will print the k. Now we can see we have created just for playing only 3 threads by means of 3 classes class A class B and class C they are the from Thread class the run method is created. Now once these threads are created now it is our turn that how this main class can be composed. So, that all these three threads can be executed.

(Refer Slide Time: 24:12)



The slide is titled "Thread priority : An example" and features a Java code snippet for a class named ThreadPriority. The code defines a main method that creates three threads: threadA, threadB, and threadC. ThreadC is assigned the highest priority (Thread.MAX\_PRIORITY), threadB is assigned a priority one higher than threadA (threadA.getPriority()+1), and threadA is assigned the lowest priority (Thread.MIN\_PRIORITY). The threads are then started, and the main thread prints "End of main thread".

```
class ThreadPriority
{
    public static void main (String args[])
    {
        A threadA=new A();
        B threadB=new B();
        C threadC=new C();
        threadC.setPriority (Thread.MAX_PRIORITY);
        threadB.setPriority (threadA.getPriority()+1);
        threadA.setPriority (Thread.MIN_PRIORITY);
        System.out.println ("Started Thread A");
        threadA.start();
        System.out.println ("Started Thread B");
        threadB.start();
        System.out.println ("Started Thread C");
        threadC.start();
        System.out.println ("End of main thread");
    }
}
```

The slide also includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a man in the bottom right corner.

So, here is the main method as we see here this is the main methods and here we see we have created 3 threads namely threadA threadB and threadC of the class A B and C that we have discussed.

Now, once the Threads is created here we see we assign the Priority of the 3 threads like threadC setPriority Thread MAX PRIORITY and then threadB setPriority threadA getPriority+1, what is a Priority that the threadA Priority is there that by 1. So, this is the Thread Priority sets and similarly threadB threadA Priority also Thread MIN PRIORITY. So, these are the different ways the Priority can be set for some different purposes we have setting this one. So, Thread Priority can be set this way, once the Priority set we just simply start the execution of this thread and as you see this is basically starting and start.

Now as you see whenever the threads are started it is not started in this way such this is the first A will start B will start C will start is not that, if you see the print statement and then in our demonstration we will see how this program gives the print output then we will see that here out of these which has the first highest priority. For example, here threadC has the highest Priority it will be executed first, then we see the threadB is the next Priority because of threadA the lowest priority.

So, threadB will be executed and then ThreadA and then then the loop will be in an inter min low a depending on the Priority the different execution will take place. So, this way

the ThreadPriority can be set and then it basically used by the scheduler to schedule the Thread in its own way

(Refer Slide Time: 26:10)

```
public class Test1 {  
    static void main(String[] args){  
        Thread t1 = new Thread(new R(1));  
        Thread t2 = new Thread(new R(2));  
        t1.start();  
        t2.start();  
        try {  
            t1.join(); // waits until t1 has terminated  
            t2.join(); // waits until t2 has terminated  
        }  
        catch (InterruptedException e){ }  
        System.out.println("done");  
    }  
}
```

Now, here is another example of the join method we have discussed, as I told you it is also considered the con. So, if suppose one Thread cannot begin if the other Threads are not finished its execution. So, in that case the other Threads switch should be waited for to begin another Threads that can be controlled using join method here and this is a very simple example as we see suppose t 1 and t 2 are the Threads are created, then we create t 1 is a Thread and t 2 is another Thread by means of Thread class somehow created there and we create by some methods are Runnable interface that means ok.

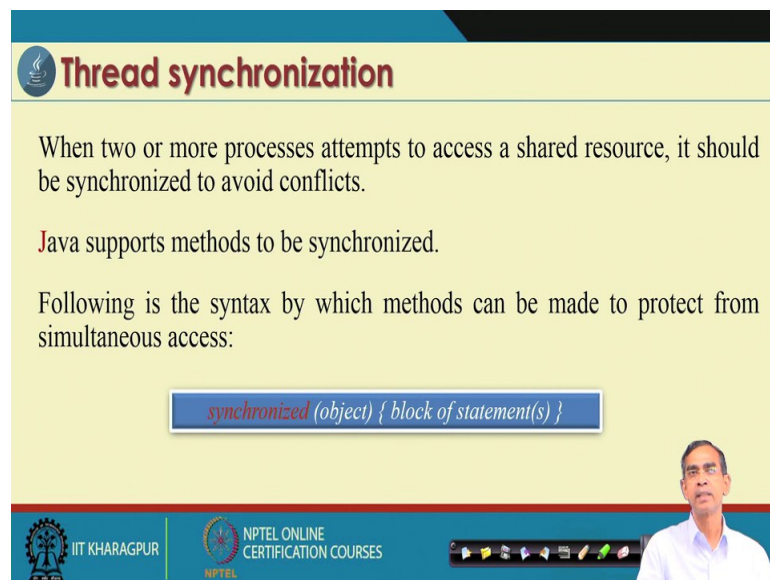
So, we have to run implement the two-run method by implementing the Runnable interface that can be passed. So, the t 1 and t 2 the 2 Thread object can be created and then they can be started its execution by calling t 1 start. So, what the thing is that this 2 Thread will start its execution because of these and then here try-catch there is a t 1 join t 2 join. So, there is a t 1 Thread t 2 Thread and then here t 1 join t 2 join so, it will wait till all the Threads are finished.

So, then this one then only it will come to this one then it will execute the final statement. So, join is basically the idea is that if there are many threads are there if we join. So, all the. So, here the Thread point execution will wait until all threads are finished its execution come here then the next task will take place there. So, this is the

concept of join is basically communicating or controlling the inter-process threads actually.

Now, our last topic so, regarding this multithreading is basically synchronization of threads, now the synchronization of threads in means that if two or more threads enter into the same process then it may lead to lot of abnormality. So, what we can do is that, if we use the synchronizations that automatically the Java run time manager will control their executions say that no 2 Thread should not access the same data or same section together that is called the critical section actually those are the students from computer science they probably know that critical section is semaphore implementation.

(Refer Slide Time: 28:38)



**Thread synchronization**

When two or more processes attempts to access a shared resource, it should be synchronized to avoid conflicts.

Java supports methods to be synchronized.

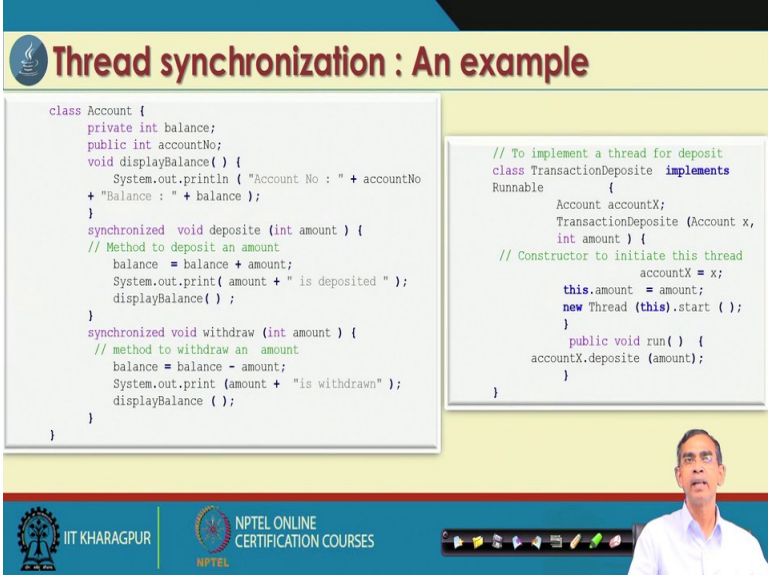
Following is the syntax by which methods can be made to protect from simultaneous access:

```
synchronized (object) { block of statement(s) }
```

The slide features a blue header with the title 'Thread synchronization' and a small icon of a flame. The main content is on a yellow background. At the bottom, there is a blue footer containing the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video feed of a man in a white shirt.

So, it is basically synchronization, it is very simple the methods which we want to synchronize just use the synchronize keyword before them and then synchronization automatically takes care it is so, easy so, for the implementation is concerned.

(Refer Slide Time: 28:54).



### Thread synchronization : An example

```
class Account {
    private int balance;
    public int accountNo;
    void displayBalance() {
        System.out.println ( "Account No : " + accountNo
        + "Balance : " + balance );
    }
    synchronized void deposit (int amount ) {
        // Method to deposit an amount
        balance = balance + amount;
        System.out.print( amount + " is deposited " );
        displayBalance() ;
    }
    synchronized void withdraw (int amount ) {
        // method to withdraw an amount
        balance = balance - amount;
        System.out.print (amount + " is withdrawn" );
        displayBalance () ;
    }
}
```

```
// To implement a thread for deposit
class TransactionDeposit implements
Runnable
{
    Account accountX;
    TransactionDeposit (Account x,
    int amount ) {
        // Constructor to initiate this thread
        accountX = x;
        this.amount = amount;
        new Thread (this).start () ;
    }
    public void run() {
        accountX.deposit (amount);
    }
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, here let us have a very simple example that how the synchronization is there. Now in banking transaction say suppose one Account where from the 2; one from the ATM and one from the Bank terminal want to access the same account sometimes, in one account one transaction one wants to use withdraw and another is to deposit.

Now, if they are not synchronized properly then, in fact, it can leads to a different result. For example, say suppose in the account balance is 1000 and then once a deposit is 500 and withdraw is 300, if it is not successful then what is the out outcome will be that either it is ok. So, it is basically the correct output should be 1200 because 500 deposits and 300 withdraw, but if it is not successful then you may get that output as either 1500 or 700 because of the abnormality the situation is there. This means that both the execution transaction deposit and withdraw should not take place simultaneously. Once if it is not finished then others should not be executed like this one.

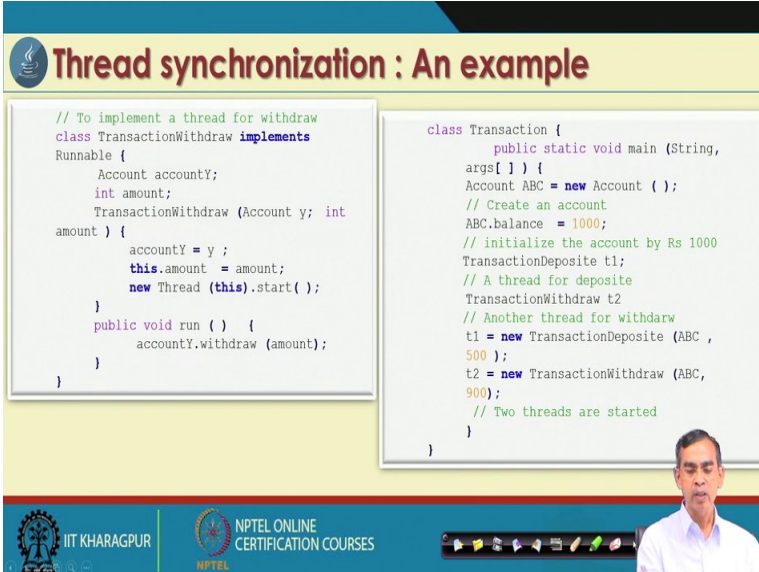
So, this is the one example that to this one here basically one class Account that is a normal class which has some members and methods are there basically balance, account number and displayBalance and whatever it is there is a very simple method are there. And in this Account we define two methods deposit and withdraw and the deposit is basically depositing the money. So, balance whatever it is their amount it will be adjusted and withdraw means it will just this one.



Now, here we can see we use the keywords synchronized here one this means that if the two methods are called for this class object account then they are basically already called in a synchronized manner. So, it is right. So, this class is very simple to understand and then only the thing that you should use the synchronize keyword here now.

So, this is the idea about and here is basically the class transaction we can say the transaction is basically implementation using a threading example here we can see, we implement transaction deposit there is two methods two transaction deposit class one is related deposit and it basically implements Runnable. So, the run method as we declared here this is basically the Constructor of this class and then run method is basically right account X dot deposit basically these run method call this deposit method. So, this is the TransactionDeposit.

(Refer Slide Time: 31:39)



The slide is titled "Thread synchronization : An example". It contains two code snippets in Java. The first snippet shows the implementation of the TransactionWithdraw class, which implements the Runnable interface. It has a constructor that takes an Account object and an amount, and a run method that calls the withdraw method on the account object. The second snippet shows the Transaction class with a static main method. The main method creates an Account object, initializes its balance to 1000, creates a TransactionDeposit object (t1) and a TransactionWithdraw object (t2), and starts both threads. The slide also features the IIT Kharagpur logo and NPTEL Online Certification Courses branding at the bottom.

```
// To implement a thread for withdraw
class TransactionWithdraw implements
Runnable {
    Account accountY;
    int amount;
    TransactionWithdraw (Account y; int
amount ) {
        accountY = y ;
        this.amount = amount;
        new Thread (this).start() ;
    }
    public void run () {
        accountY.withdraw (amount);
    }
}

class Transaction {
    public static void main (String,
args[ ] ) {
        Account ABC = new Account ( );
        // Create an account
        ABC.balance = 1000;
        // initialize the account by Rs 1000
        TransactionDeposit t1;
        // A thread for deposit
        TransactionWithdraw t2
        // Another thread for withdraw
        t1 = new TransactionDeposit (ABC ,
500 );
        t2 = new TransactionWithdraw (ABC,
900);
        // Two threads are started
    }
}
```

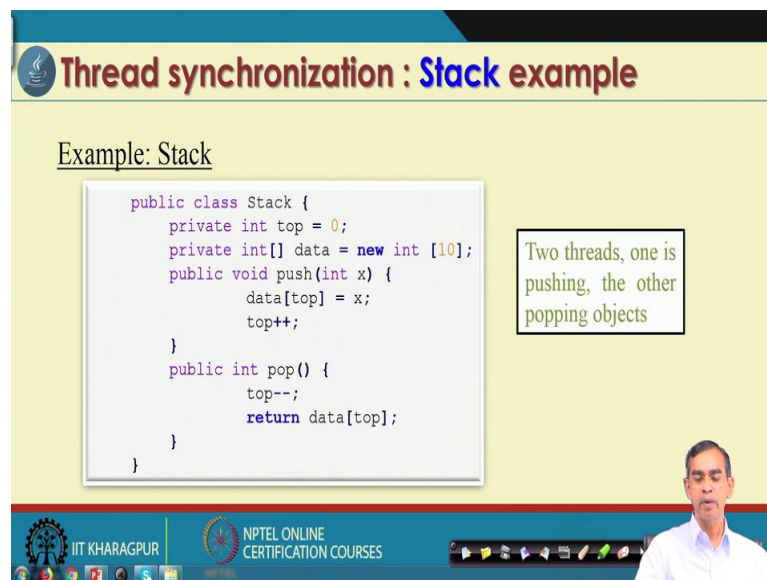
And similarly, the transaction withdraw also another method as we have declared here transaction withdraw it is a same the Runnable interface and this is a constructor for this and here is the run method as you see is basically called the withdraw method of these object account y.

Now, so, the two methods withdraw and deposit being synchronized they are basically operated for the two operations, nowhere is the methods the main method as we see. So, so main method this is the main method as we see the transaction is the name of the class and the main method it is here its basically for the account class we create an object ABC

and then that balance is 1000 and then we call this Thread t 1 TransactionDeposit TransactionWithdraw these are the two threads are executed in parallel t 1 and t 2 are the two threads; that means, here we want to say imply that two executions are invoked at the same time and then if there is no you can run this program with removing synchronized what will happen.

And if you run this program with synchronizing what will happen you can see readily the difference, but difference may not be always because the data race example that we have discussed because of this things it may leads to abnormal results because if it is not there, but if you synchronize, it always give the correct result all the time.

(Refer Slide Time: 33:08)



The slide is titled "Thread synchronization : Stack example". It contains a code block for a Java Stack class and a text box with a note.

```
public class Stack {  
    private int top = 0;  
    private int[] data = new int [10];  
    public void push(int x) {  
        data[top] = x;  
        top++;  
    }  
    public int pop() {  
        top--;  
        return data[top];  
    }  
}
```

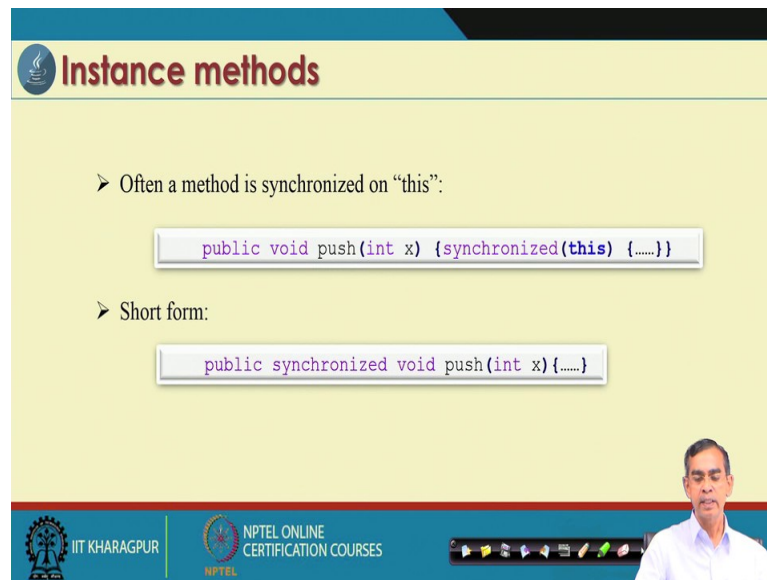
Two threads, one is pushing, the other popping objects

The slide also features the IIT KHARAGPUR logo and NPTEL ONLINE CERTIFICATION COURSES text at the bottom.

So, this is the idea about the synchronized and then again in the stack implementation the same synchronize also we can used it anyway. So, that stack is basically the implementation in a data structure that is there. So, I just want to avoid this discussion here we will discuss this thing when we will discuss about the in our demonstration who will have more examples and then more execution so, that we can run it.



(Refer Slide Time: 33:38)



### Instance methods

- Often a method is synchronized on "this":

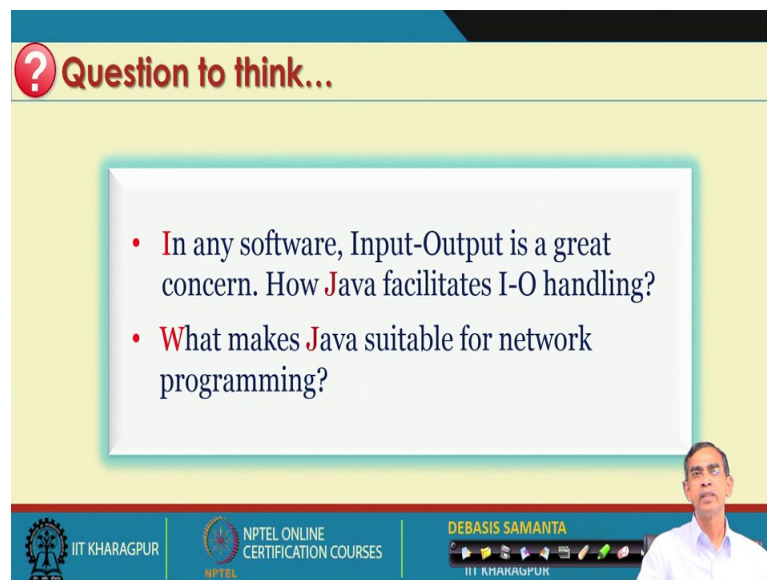
```
public void push(int x) {synchronized(this) {.....}}
```
- Short form:

```
public synchronized void push(int x) {.....}
```

The slide features a yellow background with a blue header. At the bottom, there are logos for IIT Kharagpur and NPTEL, along with a video feed of a man in a white shirt.

And so, this is the synchronization that we can have it and using the synchronized word that, we can control and then process the Thread execution there.

(Refer Slide Time: 33:42)



### ? Question to think...

- In any software, Input-Output is a great concern. How Java facilitates I-O handling?
- What makes Java suitable for network programming?

The slide has a yellow background with a blue header. At the bottom, there are logos for IIT Kharagpur and NPTEL, along with a video feed of a man in a white shirt.

So, anyway so, this is about multithreading the concept is very vast and many more things are to be discussed in detail anyway. So, although we have used the two modules many more things should be discussed while we will have a quick I mean complete demonstration on the different aspects of multithreading programming.

So, this is about the multithreading and after multithreading, the next thing is very important about that how the input and output way program can be handled because Java is for internet programming and then input and output from the many sources. So, how Java can manage the different sources of inputs and then so, output also we will discuss in our next discussion.

Thank you.