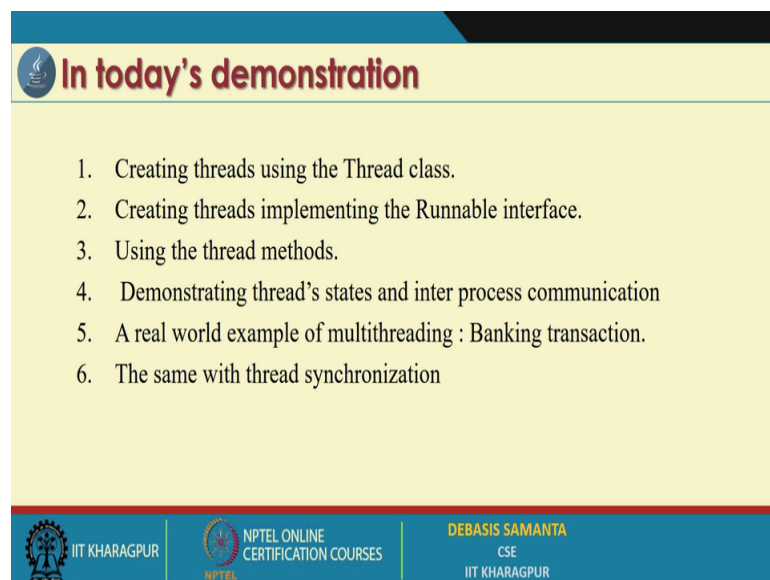


Programming in Java
Prof. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 29
Demonstration – XI

So, today we have planned our demonstration based on the lessons that we have learned in the last 2 modules on multithreading in Java. So, as you know we have discussed about multithreading concepts and here basically we have discussed about how the threads can be created and then the process communication among the different threads, the states of different threads and finally the synchronization aspects.

(Refer Slide Time: 00:35)



In today's demonstration

1. Creating threads using the Thread class.
2. Creating threads implementing the Runnable interface.
3. Using the thread methods.
4. Demonstrating thread's states and inter process communication
5. A real world example of multithreading : Banking transaction.
6. The same with thread synchronization

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA
CSE
IIT KHARAGPUR

So, in today's demo we will learn everything from the practical point of view, we will see exactly how thread can be created and then those thread can be executed.

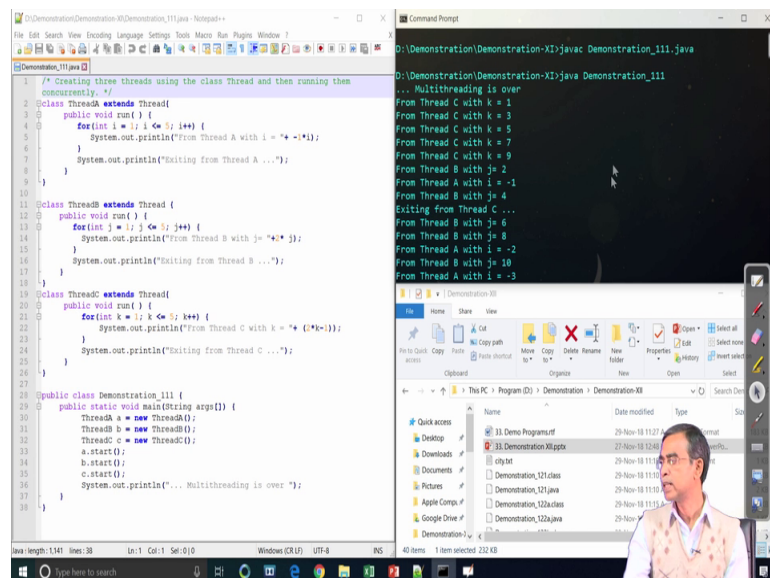
And then we have already discussed that there are 2 ways of creating thread. So, using thread class and runnable interface. So, we will learn about this the 2 things the 2 ways of creating the threads and running them and there are different methods to control the threads. So, there will be sufficient demonstrations on them. So, that we can learn about what are different methods to control the threads and then for the inter process communication again, there are few methods we will see exactly how this inter process

communicating methods can be invoked for the threads and then again the thread can be controlled.

Now finally, we will discuss about one real life example is very small example. So banking transaction so how the transactions in a bank right for the operation of an account holder can be managed and that is the using threads itself. So, this is basically by the process of synchronization. So, we will discuss about this things.

So, let us have the demo on these aspects. So, first we have let us have the demo on how we can create a thread using thread class.

(Refer Slide Time: 02:15)



The screenshot shows a Java IDE on the left and a Command Prompt on the right. The IDE displays the source code for a Java program named 'Demonstration_111.java'. The code defines three classes: ThreadA, ThreadB, and ThreadC, all extending the Thread class. ThreadA's run method prints 'From Thread A with i = *-*-' for i from 1 to 5. ThreadB's run method prints 'From Thread B with j = *-*-' for j from 1 to 6. ThreadC's run method prints 'From Thread C with k = *-*-' for k from 1 to 5. The main method in the 'Demonstration_111' class creates instances of these three threads and starts them. The Command Prompt shows the output of the program, which is interleaved, demonstrating concurrent execution. The output starts with 'Multithreading is over' and then shows the execution of Thread C (k=1 to 5), Thread B (j=1 to 6), and Thread A (i=1 to 5) in a non-sequential order.

So, in this program as we see we see here this program if you see here we have created 3 threads namely here, we have created 3 threads as we see threads A and this is basically extends thread A, this means this basically threads using the class threads and so creating a thread means, you have to override the run method and as we see we have overriding the run method here and this run method is basically as we see from this code this run method will run for 0 to 5 loops and in each loop it will print the negative number.

So, starting from minus 1, 2, 5 like and next thread. So, it is a thread B again it is extend the thread class and here also we override the run method these also loops for 0 to 5 6 times and in this loop as we see these method will print the 6, the first 6 even numbers and similarly another thread, thread C it is also a extension of class thread and then here

the run method which is basically a simple loop here and then loop will run from k equals to 1 to k that mean 5 times and here it will print first 5 odd numbers.

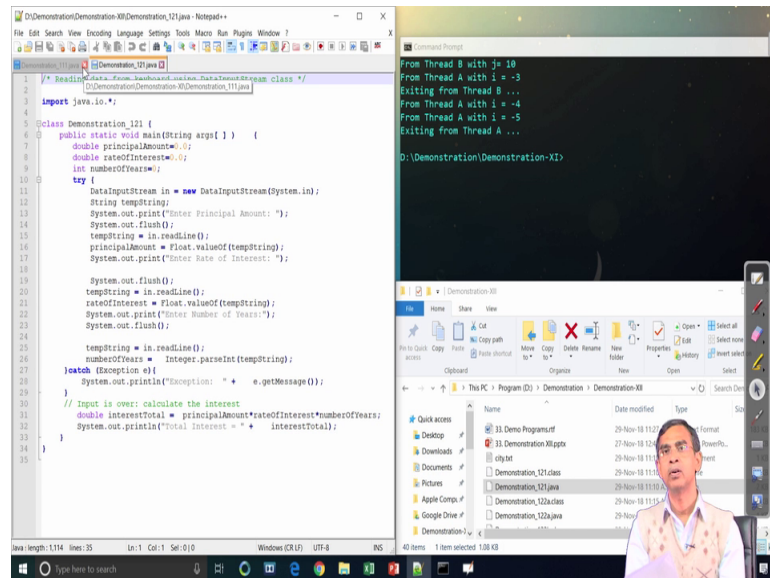
Now so these are the 3 threads. So, our plan is to run these 3 threads simultaneously now here is the main method and in this main method, we see how we can create the 3 thread objects A, B and C these are the 3 thread objects for the 3 classes like thread A, thread B and thread C. Now this is the syntax as you see how we can create the 3 thread object for the 3 thread classes that we have defined here. Now in order to run the thread so we have to call the method start for each thread object for example, a dot start this means that thread A will run b dot start that this will run c dot start this will run.

So here, in this main method as we see so these are the 3 threads we will run, but in addition to the main method itself is a thread. So, all together here actually 4 thread will run. So, this is the thread a will run this means this thread will print the negative numbers thread B when it will run it will print odd even numbers thread C will run when it will print the odd numbers and here this main thread other than this executing or invoking this threads, it will also print this statement and we are not sure the which thread will be executed in which order because here, the scheduler will take all the threads and depending on it is own because, here no priority has been assigned actually all threads are in random pattern, it will be executed. Now so this is the program.

Now, we can have some idea about how this program, if it run then how it will give us the output. So, name of the program and we have given here demonstration underscore 11 1 ok. So, this program is now compiled and then we are running this program and yeah as we see these program, it gives the output here multithreading is over this is basically from the main thread and here you see thread C thread B A B B they are basically in random order somehow thread C which was (Refer Time: 06:05) invoked in the main method last, but it got executed first and then B A B in inter living manner and we see all thread when they are executed they are printing their number in that order of course, in the way the loop will executed.

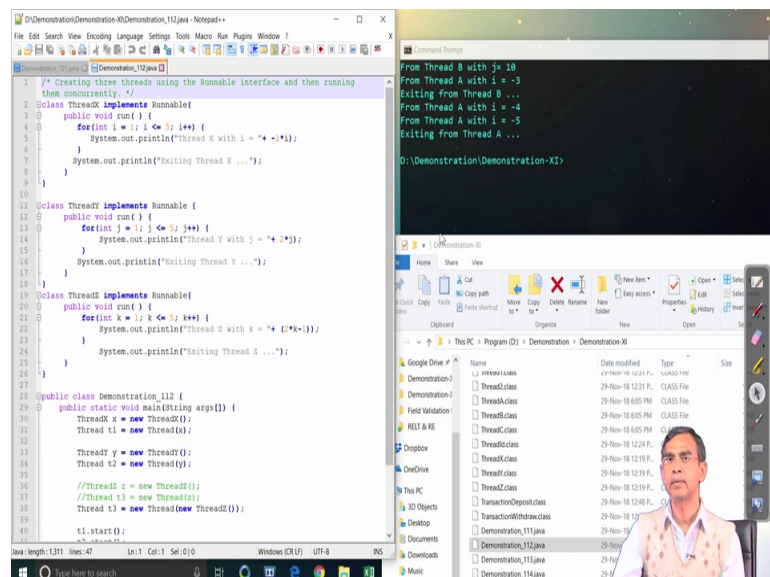
So, this basically example gives that how the different threads will be executed and that is the execution of the 3 threads. Now let us have another demo.

(Refer Slide Time: 06:35)



So, in this example we have created 3 threads using class thread that is the extends thread now there is an another way of creating threads and running them using the runnable interface. So, this is the program which basically gives us an idea about how the same problem, which we have implemented using thread class can be also implemented using runnable interface 11.2 ok.

(Refer Slide Time: 07:09)



Now so this is the example what we can see here how the thread can be executed using runnable interface. Now let us see we define thread X the another thread which

implements runnable interface. So, implements runnable; that means, we want to create the thread using runnable interface.

In runnable interface there is an abstract method, which is basically run. So here, basically implementing means we have to override the run method. So, this is the code that basically implements the run method and this implementation is same as the thread A extends thread class, in the last example similarly thread Y is the same version of thread B and this is the implementation of run method for this thread X and similarly this is the implementation of run method for thread Z, it basically implementing runnable interface.

So, as we see there are 3 threads we have created using runnable interface. Now let us see how we can create the. So, these are the basically class declaration for the 3 threads and now this is a main method, this method is basically creates the 3 threads as we defined thread X thread Y and thread Z. Now here, you see first we create a X, it is basically the object of the class thread X and then we create a thread of that objects. So, this basically t a t 1 is the thread object for the class thread X.

So, to use it using runnable interface method we have to have this syntax like thread t 1 that mean t 1 is a thread and we pass the thread class that is the x object, which is basically implementation runnable interface actually and then so these basically, create the thread t 1 for the thread X likewise, these basically t 2 is the thread 2 for the class object y and thread thread Y actually and this is the another way short cut method also we can use.

So, t 3 is the thread object new thread and in this constructor. So, there are different constructor as we see to create the thread, this is the one constructor where you have to pass the object of the thread, but here we can pass the constructor directly. So, this is the another way that the thread object can be the thread object can be created. So, these are one way of creating (Refer Time: 09:51) actually anyway either these way or these way you can do whatever it is there.

Now so we have created 3 thread t 1, t 2 and t 3 and now we are in a position to run them.

(Refer Slide Time: 10:03)

```
11 class ThreadX implements Runnable {
12     public void run() {
13         for(int i = 1; i <= 5; i++) {
14             System.out.println("Thread X with i = * * * * *");
15         }
16         System.out.println("Exiting Thread X ...");
17     }
18 }
19 class ThreadY implements Runnable{
20     public void run() {
21         for(int j = 1; j <= 5; j++) {
22             System.out.println("Thread Y with k = * * * * *");
23         }
24         System.out.println("Exiting Thread Y ...");
25     }
26 }
27
28 public class Demonstration_112 {
29     public static void main(String args[]) {
30         ThreadX x = new ThreadX();
31         Thread t1 = new Thread(x);
32
33         ThreadY y = new ThreadY();
34         Thread t2 = new Thread(y);
35
36         //Thread z = new ThreadZ();
37         //Thread t3 = new Thread(z);
38         Thread t3 = new Thread(new ThreadZ());
39
40         t1.start();
41         t2.start();
42         t3.start();
43
44         System.out.println("... Multithreading is over ");
45     }
46 }
47
```

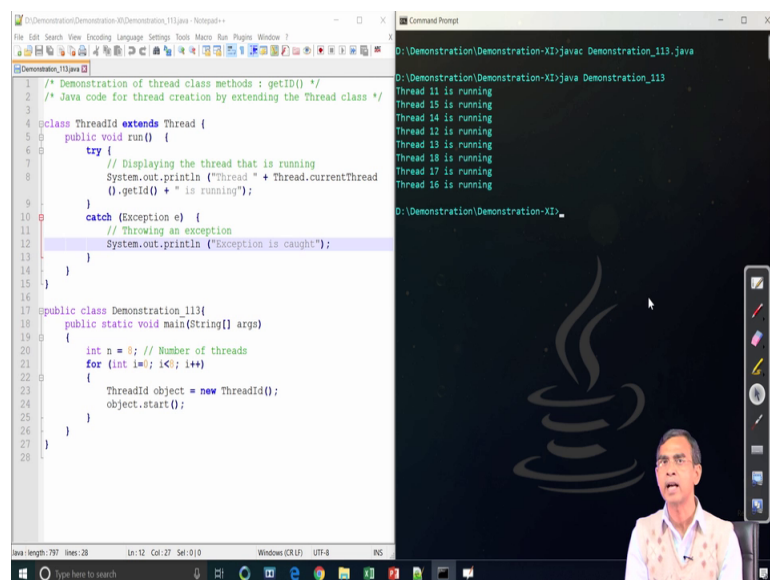
So here again, we run t 1 start, t 2 start, t 3 start means 3 threads are now started invoked for they are running and then this is basically the statement which is basically in the main statement basically, the same program that we have done using thread class, but here only the difference is that we have done the same thing using runnable interface and definitely it is basically same program, but implementation is different definitely they will give the same output. Now let us see the output of this program ok.

(Refer Slide Time: 10:39)

```
Thread X with i = -2
Thread X with i = -3
Thread X with i = -4
Thread X with i = -5
Thread Z with k = 1
Thread Z with k = 3
Thread Z with k = 5
Thread Z with k = 7
Thread Y with j = 2
Thread Y with j = 9
Exiting Thread X ...
Exiting Thread Z ...
Thread Y with j = 4
Thread Y with j = 6
Thread Y with j = 8
Thread Y with j = 10
Exiting Thread Y ...
... Multithreading is over
```

So, this thread as you see X Z Y Z all this things run in an inter living manner, because whenever the thread is executed they are basically executing parallelly, but here in our display we are our producing the output synchronizing. So, better can be that if we have the three display unit for each display the different thread can produce their output then we can see that how the 3 threads are running independently rather, but here is basically we are producing the output from the 3 threads on the same input same output screen. So, that is why we are getting the output in the inter live manner. So, this is the way of creating threads, we have learn about we can create threads in 2 ways using the class thread and the runnable interface.

(Refer Slide Time: 11:29)



The screenshot shows a Java IDE on the left and a terminal window on the right. The IDE displays the following code:

```
1 /* Demonstration of thread class methods : getId() */
2 /* Java code for thread creation by extending the Thread class */
3
4 class ThreadId extends Thread {
5     public void run() {
6         try {
7             // Displaying the thread that is running
8             System.out.println("Thread " + Thread.currentThread
9                 ().getId() + " is running");
10        }
11        catch (Exception e) {
12            // Throwing an exception
13            System.out.println("Exception is caught");
14        }
15    }
16
17 public class Demonstration_113{
18     public static void main(String[] args)
19     {
20         int n = 8; // Number of threads
21         for (int i=0; i<n; i++)
22         {
23             ThreadId object = new ThreadId();
24             object.start();
25         }
26     }
27 }
28
```

The terminal window on the right shows the output of the program, listing thread IDs and their status:

```
D:\Demonstration\Demonstration>java Demonstration_113
ThreadId 11 is running
ThreadId 15 is running
ThreadId 14 is running
ThreadId 12 is running
ThreadId 13 is running
ThreadId 18 is running
ThreadId 17 is running
ThreadId 16 is running
```

Now, our next illustration is basically to every threads, whenever it is in execution from the program point of view we can get have information about the threads by means of their identity. So, the I d of a thread can be accessed by a method called the get I d, which is defined in the class thread in the Java dot lang dot methods class. Now here, let us see one example as we see here now. So, these basically this is the name of the class that we are going to declare is the thread I d thread I d is basically extend thread always whenever you have to create thread class you have to have an extension or either thread class or implements runnable interface that is the basic concept of course,.

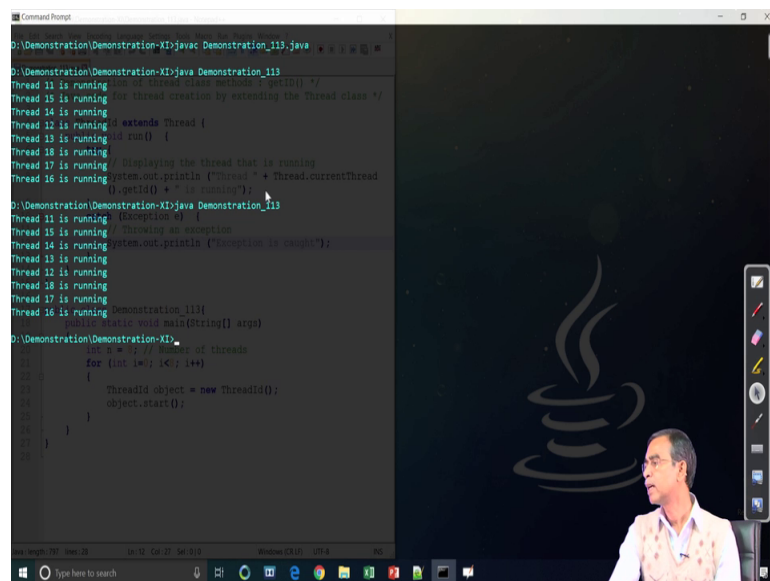
Now here, the run method basically thread has it is own abstract run method, but whenever you have to extend it we have to basically implement override this run method.

Now this run method is very simple here basically it include one print statement which print the thread dot current thread the; that means, whichever the thread is in execution it will basically for that thread and then dot get I d means for that thread the I d and it basically it will give the things that which is a current thread is in execution and corresponding to that thread, what is the I d will be displayed and print and so this is basically, under try and catch exception to handle the exceptions if any things occurs whenever this program is in execution.

Now here is the main method here, we can see in this main method there will be in fact, i equals to 0 to 8; that means, 9 9 threads will be executed we are basically running for the same class thread I d, but 9 occasions. So, the 9 thread will be executed and here although we are calling in this way, but whenever these threads are in execution they will be executed simultaneously in parallel.

So, concurrently so these basically, it creates a thread in each loop and then that thread is basically executes. So, this actually these way the nine loops will be created 9 threads will be created and then each thread will be executed. So, this is the basically idea of the program and in this program as we see the output this program will see you will display the output whenever is thread is in execution and corresponding to their I d.

(Refer Slide Time: 14:07)



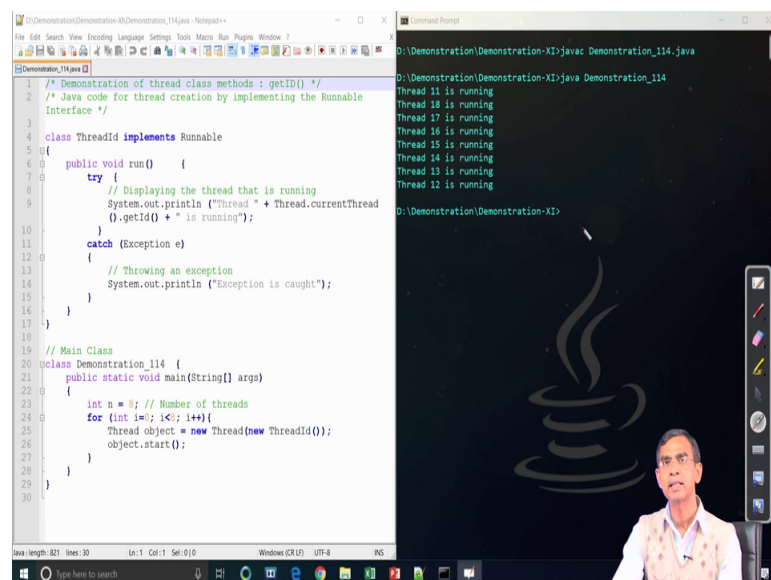
```
Command Prompt
D:\Demonstration\Demonstration-XI>javac Demonstration_113.java
D:\Demonstration\Demonstration-XI>java Demonstration_113
Thread 11 is running
Thread 15 is running
Thread 14 is running
Thread 12 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
D:\Demonstration\Demonstration-XI>java Demonstration_113
Thread 11 is running
Thread 15 is running
Thread 14 is running
Thread 13 is running
Thread 12 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
D:\Demonstration\Demonstration-XI>
public class Demonstration_113 {
    public static void main(String[] args) {
        for (int i = 0; i < 9; i++) {
            ThreadID object = new ThreadID(i);
            object.start();
        }
    }
}
```

So, as we see here thread that is I 1 is running thread I 5 is running I 4 is running now at we can see 1 to 9. So, 9 threads are basically in execution. So, this way we can see the

threads are executing in parallel, but it is not sure if we run this program again not necessarily in the same output will be let us run the same program again as we see possibly it will give you the different ordering of the execution as we see here in this case; obviously, in the but here in the last 3 we can see the different ordering. So, ordering is in fact, provably stick or it is not predictable rather unpredictable ordering actually anyway. So, this is the idea about as we see is a simple example where the, I d of the thread can be can be access whenever a thread is in execution.

Now, our next execution next program basically the same, but using runnable interface.

(Refer Slide Time: 15:07)



The screenshot shows a Java IDE on the left and a terminal window on the right. The IDE displays the following code:

```
1  /* Demonstration of thread class methods : getId() */
2  /* Java code for thread creation by implementing the Runnable
3  Interface */
4  class ThreadId implements Runnable
5  {
6  public void run() {
7  try {
8  // Displaying the thread that is running
9  System.out.println ("Thread " + Thread.currentThread
10 () .getId() + " is running");
11 }
12 catch (Exception e)
13 {
14 // Throwing an exception
15 System.out.println ("Exception is caught");
16 }
17 }
18 }
19 // Main Class
20 class Demonstration_114 {
21 public static void main(String[] args)
22 {
23 int n = 8; // Number of threads
24 for (int i=0; i<=; i++){
25 Thread object = new Thread(new ThreadId());
26 object.start();
27 }
28 }
29 }
30 }
```

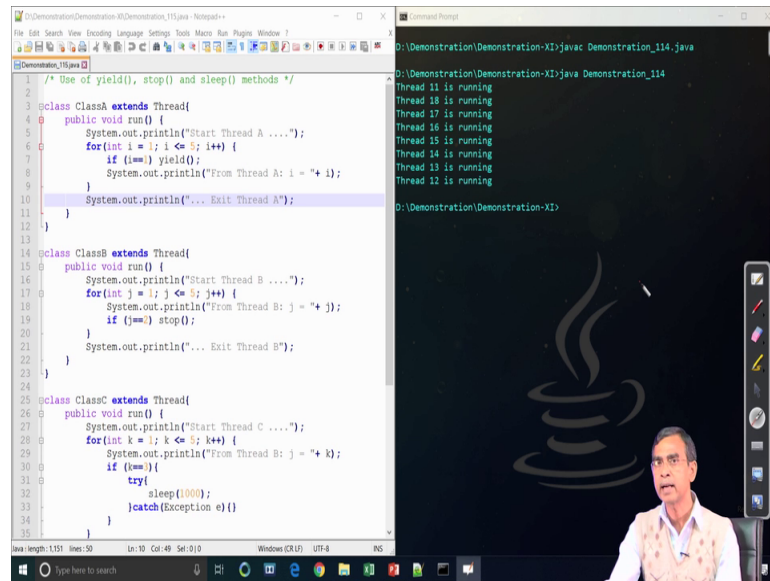
The terminal window shows the output of the program, which is a list of thread IDs and their status, such as "Thread 11 is running", "Thread 18 is running", etc., demonstrating the non-deterministic order of execution.

As we have seen in the last program, we have created the thread class, you have we have extended the thread class again the same program as we see here using the runnable interface same code, but only the implement this is a different and here again as usual. So, this is also different because in order to execute or in order to create a thread which has been implemented by runnable interface. So, this is basically the structure of syntax that we have to follow and then we have to just as in the thread class we have to just invoke the start method for each thread to run it.

So, these will create one object each time the loop will roll and then it basically start again the same output similar output as in the previous this illustration we will be able to see it. So, here is basically we are running the program here threaded. So, we can see the similar kind of output as we can see you can see the bigger screen so, that we can see it

clearly? Yes ok. So, this is the basically output as we see. So, here basically in the different order as we see this is basically same program same logic, but that logic is implemented using implement using runnable interface ok. Now our next illustration to express, the different states of a thread and as we know the different control by which the state of a thread can be managed.

(Refer Slide Time: 16:51)



```
1 /* Use of yield(), stop() and sleep() methods */
2
3 class ClassA extends Thread{
4     public void run() {
5         System.out.println("Start Thread A ...");
6         for(int i = 1; i <= 5; i++) {
7             if (i==1) yield();
8             System.out.println("From Thread A: i = "+ i);
9         }
10        System.out.println("... Exit Thread A");
11    }
12 }
13
14 class ClassB extends Thread{
15     public void run() {
16         System.out.println("Start Thread B ...");
17         for(int j = 1; j <= 5; j++) {
18             System.out.println("From Thread B: j = "+ j);
19             if (j==1) stop();
20         }
21         System.out.println("... Exit Thread B");
22     }
23 }
24
25 class ClassC extends Thread{
26     public void run() {
27         System.out.println("Start Thread C ...");
28         for(int k = 1; k <= 5; k++) {
29             System.out.println("From Thread C: k = "+ k);
30             if (k==1){
31                 try{
32                     sleep(1000);
33                 }catch(Exception e){}
34             }
35         }
36     }
37 }
```

```
D:\Demonstration\Demonstration>javac Demonstration_114.java
D:\Demonstration\Demonstration>java Demonstration_114
Thread 11 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
Thread 15 is running
Thread 14 is running
Thread 13 is running
Thread 12 is running
D:\Demonstration\Demonstration>
```

For example, there is yield method, stop method, suspend method, resume method, sleep method, whatever it is there.

Now in this example, we will just see exactly how the sleep method will work for us and then stop method also we will see yield stop and then sleep these are the 3 methods we will see how this methods can be invoked for a for a current thread under execution. Now here, is the code as we see we create class A extends thread and here we again override the run method it is the simple print statement that from which state these thread in execution. So, it keeps that start from thread A and this is basically the loop here we see for i equals to 1 i less than 5. So, 1 to 5 this loop will roll and if i equals to 1, the yield method it will be there. So, then system dot out dot print from thread A i plus 1 it basically, print the value of i.

So, yield method is basically tell that this thread is in execution. Now here again class B extends thread as you see the similar structure of the program code here in this run method as well as, but here j equals to 1 to j less than 5, this run method also will loop

will roll for 5 times and here whenever j equals to 2 it basically stop; that means, these thread will be after the running up to 1 1 and 2 from 2 for the third loop one was it will basically stop and then stop for sometimes until it will stop means it is totally stop this means, these thread will no more execute; that means, these loop will although for roll for 5 times, but it will ultimately work for you only for j equals to 1 only.

Now, here again class C this is another thread we have created here similar run method k equals to 1 to 5 and here we see when k equals to 3 we call the sleep and this sleep is 30 30 1000 millisecond 1000 millisecond means one second like. So, this sleep this thread will sleep for 1 second; that means, it will roll k equals to 1 k equals to 2, whenever k equals to 3 for sleep 1 second and then again it will roll for 4 and 5.

(Refer Slide Time: 19:29)

```

17 for(int j = 1; j <= 5; j++) {
18     System.out.println("From Thread B: j = "+ j);
19     if (j==2) stop();
20 }
21 System.out.println("... Exit Thread B");
22 }
23 }
24 }
25
26 class ClassC extends Thread{
27     public void run() {
28         System.out.println("Start Thread C ...");
29         for(int k = 1; k <= 5; k++) {
30             System.out.println("From Thread B: j = "+ k);
31             if (k==3){
32                 try{
33                     sleep(1000);
34                 }catch(Exception e){}
35             }
36             System.out.println("... Exit Thread C");
37         }
38     }
39 }
40
41 class Demonstration_115{
42     public static void main (String args[]) {
43         ThreadA t1 = new ThreadA();
44         ThreadB t2 = new ThreadB();
45         ThreadC t3 = new ThreadC();
46         t1.start(); t2.start(); t3.start();
47         System.out.println("... End of execution ");
48     }
49 }
50 }

```

```

D:\Demonstration\Demonstration>javac Demonstration_114.java
Thread 11 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
Thread 15 is running
Thread 14 is running
Thread 13 is running
Thread 12 is running

D:\Demonstration\Demonstration>javac Demonstration_115.java
Note: Demonstration_115.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Demonstration\Demonstration>java Demonstration_115
... End of execution
From Thread C with k = 1
From Thread B with j= 2
From Thread A with i = -1
From Thread B with j= 4
From Thread A with i = -2
From Thread B with j= 6
From Thread A with i = -3
From Thread B with j= 8
From Thread A with i = -4
From Thread C with k = 3
From Thread A with i = -5
From Thread C with k = 5
Exiting from Thread A ...
From Thread C with k = 7
From Thread C with k = 9
Exiting from Thread C ...

```

So, these are the 3 thread classes we have created having their own mechanism and now we in the main method, we can create the 3 objects of this 3 thread classes t 1, t 2, t 3 and then we just start their execution here and then finally, one the thread is finished it basically print that end of the thread execution, but as it is here in this main method actually in parallel 4 thread threads will run concurrently. So, this is the idea about 3 threads here and here is the again output we can see how this program run fine. So, these are warning we can ignore the warning yeah.

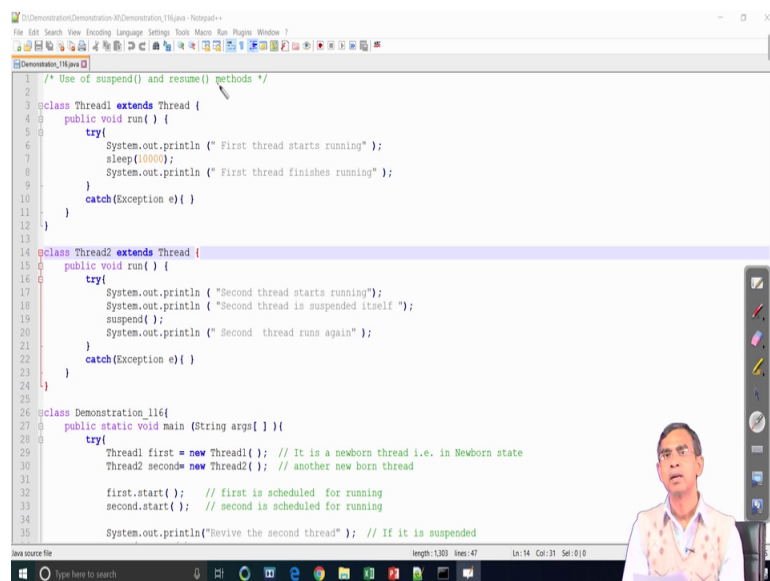
So, as we see the output seen here, here, here ok. So, end of execution somehow it is the last statement, but it is the one main thread somehow it is executed first and then thread

C, thread B, thread B in a random fashion as we see the different threads executed and also we can see the second thread, thread B actually as it is run it is basically stop for thread 1 right for the loop and for the 2 4 6, it is basically running. So, this is the way it is basically it works for us

Now, let us go to the program here the program which we are discussing. So, what essentially the multithreading basically it is there. So, as we see each thread has its own code. Now again for example, say suppose we want to run 2 methods, one method will be for sorting some array of elements and another method for searching some elements in a array. So, what we can do is that, we can create a thread for the sorting method implementation we can create another thread for implementation of searching method. So, whatever the logic there is to sort some array or elements we can write the code under the run similarly, whatever the code that is required to search an item in an array we can put into the run method.

So, this is the way that we can run our own what is called the algorithm the method and that those algorithm can be executed in parallel. So, parallel execution means that they will share the system may be in a time sharing basis or in a multi programming basis that is the part of the operating system. So, operating system; that means, here Java run time manager will manage the execution of all the threads in parallel. So, this way actually if we run says sorting and searching one by one. So, this may takes a 5 plus 5 10 seconds.

(Refer Slide Time: 22:07)



```
1  /* Use of suspend() and resume() methods */
2
3  class Thread1 extends Thread {
4  public void run() {
5      try{
6          System.out.println (" First thread starts running");
7          sleep(1000);
8          System.out.println (" First thread finishes running");
9      }
10     catch(Exception e){ }
11 }
12 }
13
14 class Thread2 extends Thread {
15 public void run() {
16     try{
17         System.out.println (" Second thread starts running");
18         System.out.println (" Second thread is suspended itself ");
19         suspend();
20         System.out.println (" Second thread runs again" );
21     }
22     catch(Exception e){ }
23 }
24 }
25
26 class Demonstration_116{
27     public static void main (String args[] ) {
28         try{
29             Thread1 first = new Thread1(); // It is a newborn thread i.e. in Newborn state
30             Thread2 second= new Thread2(); // another new born thread
31
32             first.start(); // first is scheduled for running
33             second.start(); // second is scheduled for running
34
35             System.out.println("Revive the second thread" ); // If it is suspended
```

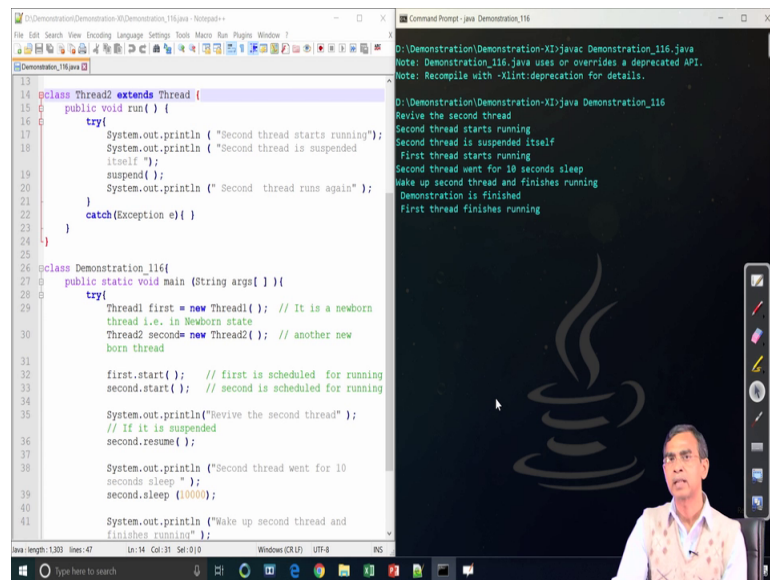
But, if you run in parallel it may take 5 seconds both the program will run, but it will the maximally utilize the CPU resources other resources, those are required in your program execution can be maximally utilized. So, this is the purpose of the thesis purpose of the threads execution.

Now, let us have the another example as we have learned the method yield stop and then sleep likewise there is some other method also like suspend and resume method. So, this examples basically explain us how the suspend and resume method work for us. So, suspend method means if you write if you can wait make a c thread wait by calling some method right then suspend method is basically ok. Any one method is running and you can call the suspend method that mean this thread will be not permanently stop, it will be temporarily withheld and the same method can be again revoked if we call the resume method.

So here is a program, which basically explain that we suspend one method thread and then the same thread can be resumed from the other thread exactly. So, here is the code. Now here, you can say thread 1, we create 1 thread and this is a run method is basically simple it is state that print a print statement that this thread starts running and then here the sleep method we call here sleep for 10 seconds and then the system dot out print l n the first thread finish running and then this is the thread 2 and here again the similar kind of code here is basically thread we call a method suspend and then the 2 threads are here. So, the threads this whenever the 2 threads are a in execution, one thread will sleep for 10 milliseconds another thread will go for suspending.

Now here in the main method, as we see we run the 3 2 threads first and second namely here and then start it and then execution. So now, let us have the code for that quickly, yeah as we just output is little bit bigger so that we can see output (Refer Time: 24:25) fine.

(Refer Slide Time: 24:27)



```
13
14 class Thread2 extends Thread {
15     public void run() {
16         try{
17             System.out.println ("Second thread starts running");
18             System.out.println ("Second thread is suspended
19             itself");
20             suspend();
21             System.out.println (" Second thread runs again" );
22         }
23         catch(Exception e){ }
24     }
25
26 class Demonstration_116{
27     public static void main (String args[] ){
28         try{
29             Thread t1 = new Thread1(); // It is a newborn
30             thread i.e. in newborn state
31             Thread2 t2 = new Thread2(); // another new
32             born thread
33
34             t1.start(); // first is scheduled for running
35             t2.start(); // second is scheduled for running
36
37             System.out.println("Revive the second thread");
38             // If it is suspended
39             t2.resume();
40
41             System.out.println ("Second thread went for 10
42             seconds sleep ");
43             t2.sleep (10000);
44
45             System.out.println ("Wake up second thread and
46             finishes running");
47         }
48     }
49 }
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

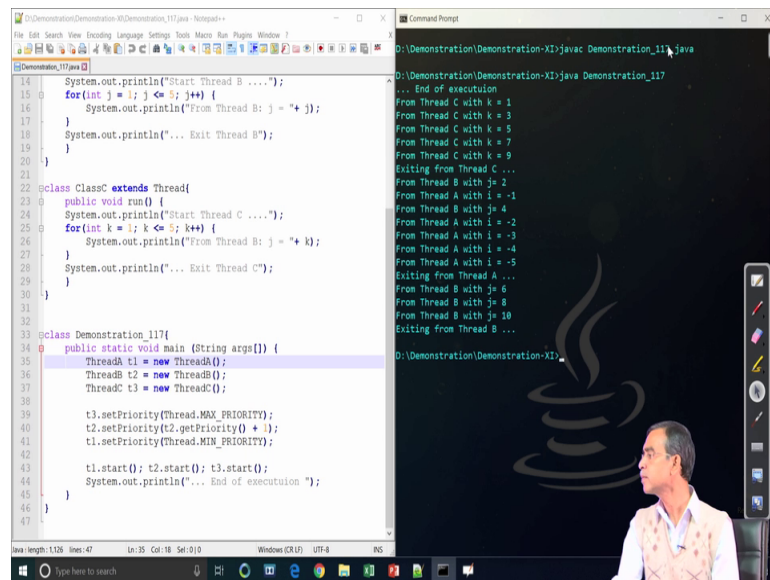
```
D:\Demonstration\Demonstration>javac Demonstration_116.java
Note: Demonstration_116.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\Demonstration\Demonstration>java Demonstration_116
Revive the second thread
Second thread starts running
Second thread is suspended itself
First thread starts running
Second thread went for 10 seconds sleep
Wake up second thread and finishes running
Demonstration is finished
First thread finishes running
```

Now here again, we see the output as we see these are the thread was in executions. So, sleep and that is why after ten second this sleep this thread actually revoked and then it produces the output like this one.

So, you can see that how the suspend and then the sleep the similarly resume method also we can call the resume method in the main method after sometime the resume method can be called so that it can resumed it now. So, this is the different method to control the thread. So, is basically mechanism the way that different thread can be controlled. Now here, we have discussed about a priority can be assigned among the threads.

(Refer Slide Time: 25:09)



```
14 System.out.println("Start Thread B ...");
15 for(int j = 1; j <= 5; j++) {
16     System.out.println("From Thread B: j = "+ j);
17 }
18 System.out.println("... Exit Thread B");
19 }
20 }
21
22 class ClassC extends Thread{
23     public void run() {
24         System.out.println("Start Thread C ...");
25         for(int k = 1; k <= 5; k++) {
26             System.out.println("From Thread B: j = "+ k);
27         }
28         System.out.println("... Exit Thread C");
29     }
30 }
31
32
33 class Demonstration_117{
34     public static void main (String args[]) {
35         ThreadA t1 = new ThreadA();
36         ThreadB t2 = new ThreadB();
37         ThreadC t3 = new ThreadC();
38
39         t3.setPriority(Thread.MAX_PRIORITY);
40         t2.setPriority(t2.getPriority() + 1);
41         t1.setPriority(Thread.MIN_PRIORITY);
42
43         t1.start(); t2.start(); t3.start();
44         System.out.println("... End of execution ");
45     }
46 }
47 }
```

```
D:\Demonstration\Demonstration>XI:java Demonstration_117
... End of execution
From Thread C with k = 1
From Thread C with k = 2
From Thread C with k = 3
From Thread C with k = 4
From Thread C with k = 5
Exiting from Thread C ...
From Thread B with j = 1
From Thread A with i = -1
From Thread B with j = 2
From Thread A with i = -2
From Thread A with i = -3
From Thread A with i = -4
From Thread A with i = -5
Exiting from Thread A ...
From Thread B with j = 3
From Thread B with j = 4
From Thread B with j = 5
Exiting from Thread B ...
D:\Demonstration\Demonstration>XI:
```

Suppose, while in the execution we can set some priority to the threads as we have discussed about there are 3 priority min priority, max priority and normal priority.

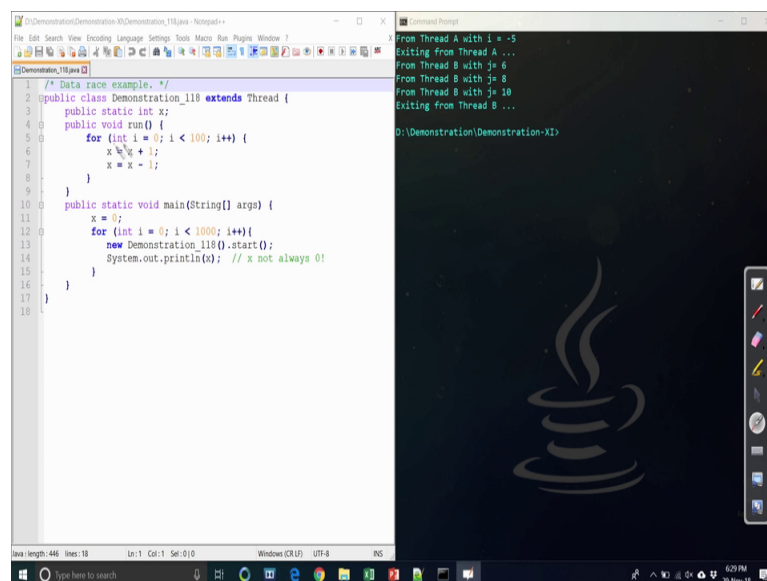
Now, the thread we can assign priority values to all these threads here now, this program will explain how we can create 3 threads and among these 3 threads we can assign the priority. So, class A is a 1 thread, this thread has a simple loop i equals to 1 to 5 and then this is the simple loop execution it will print the 3 5 numbers actually 1 to 5.

Similarly class B also similar it will print same numbers 1 to 5 and class C also same. So, both the threads are basically similar form of executions. Now here as we see in the main method as we see in the main method, let us look at the main method we created 3 3 threads t 1, t 2, t 2 we create 3 threads t 1, t 2 and t 3 here for the class thread a thread B and thread C and then here the method set priority which is defined in the class thread and then we plus argument as thread dot MAX PRIORITY, this value is already defined in the thread class.

So here, basically this basically t 3 has been assigned the maximum priority and as we see t 1 has been assigned the lowest priority and here t 2 the set priority get priority, whatever the priority of the thread at present it is basically plus 1. So, it is a random priority then 3 threads are executed invoked and then finally, the main thread it is there

Now, we will see. So, the order actually whatever it is there or you see the max priority having the thread t 3, it will be executed first. So, scheduler will know from this information that which threads needs to be executed first. As we see here because, t 3 because t 3 the thread C namely having the highest priority and as we see here basically it basically invoke first then B and then A and B they are basically random priority based on and they have been executed. So this way, we can assign the priority to a thread so that the execution of thread can be further controlled.

(Refer Slide Time: 27:53)



```
1 /* Data race example. */
2 public class Demonstration_118 extends Thread {
3     public static int x;
4     public void run() {
5         for (int i = 0; i < 100; i++) {
6             x = x + 1;
7             x = x - 1;
8         }
9     }
10    public static void main(String[] args) {
11        x = 0;
12        for (int i = 0; i < 1000; i++) {
13            new Demonstration_118().start();
14            System.out.println(x); // x not always 0!
15        }
16    }
17 }
18 }
```

```
From Thread A with i = -5
Exiting from Thread A ...
From Thread B with j= 6
From Thread B with j= 8
From Thread B with j= 10
Exiting from Thread B ...
0:\Demonstration\Demonstration-XI>
```

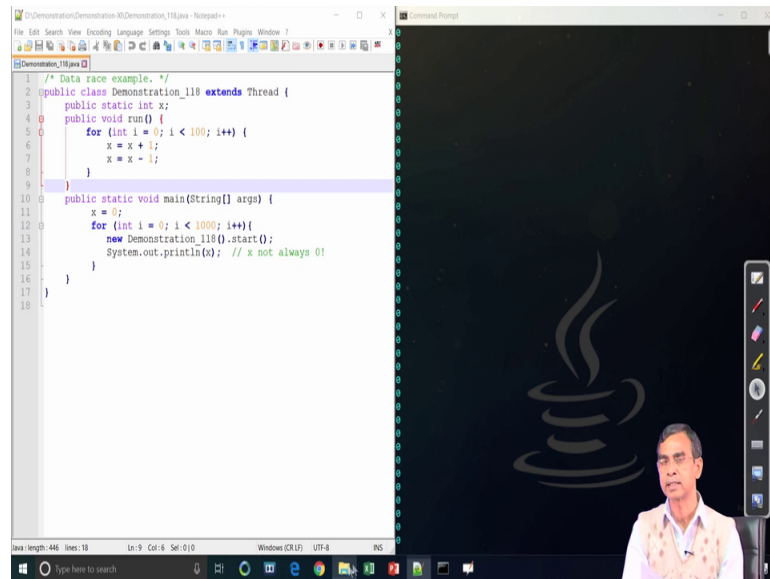
Now so here is another problem, this is very interesting problem multithreading works good usually first, but if you are not careful then you may not get the right result, now in this regard there is a problem. So, for the concurrent program execution is concern it is called the race problem, the data race problem. Now if you see the code here, now this code is very simple as we see it declare one integer as a static x and the run method is basically overriding the class method here and here we see x equals to x plus 1 x equals to x plus minus x minus 1.

Now after the end of the loop if we come here. So, what will be the desired output? Output will be 0 if x is initialized as 0 is not it, but here we see whenever we run this program it may not produce the result 0 always this is because, it is an intermittent way this statement will be executed by the scheduler in a different way and that is why it is not atomic. So, if rather non atomic the execution. So, whenever we run this main

method I mean run this loop for say here 1000 times and each time not necessarily 0 output will be produced, it can produce non 0 output as well as.

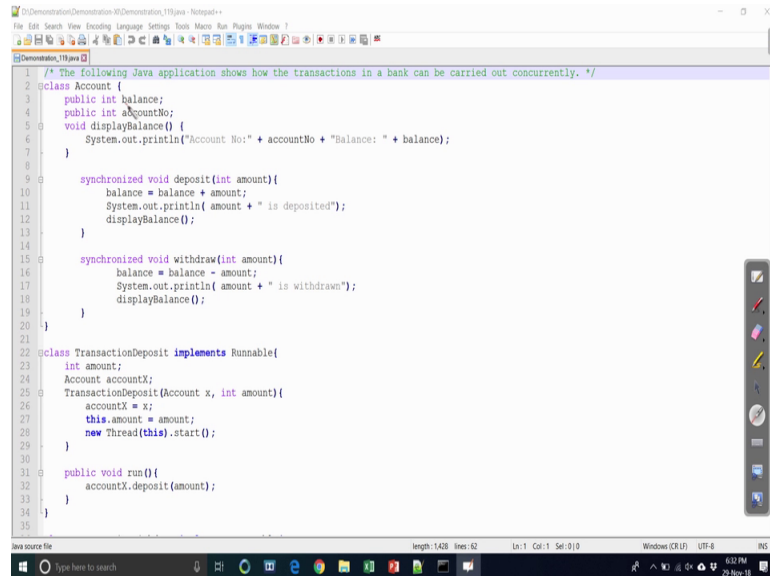
Now, let us see the execution of this program, but in the different instances if we run the different program, it will produce the different result.

(Refer Slide Time: 29:37)

The image shows a screenshot of a computer screen. On the left, a Notepad++ window displays a Java code file named 'Demonstration_118.java'. The code defines a class 'Demonstration_118' that extends 'Thread'. It has a static variable 'x' initialized to 0. The 'run()' method contains a loop that increments 'x' by 1 and then decrements it by 1, repeating this 100 times. The 'main' method starts 'x' at 0 and launches 1000 instances of 'Demonstration_118', each of which prints the current value of 'x'. On the right, a 'Command Prompt' window shows the output of the program, which consists of a vertical list of '0's, with a single '1' appearing near the top, demonstrating a data race where the final value of 'x' is not always 0.

So, if basically 1000 output has been printed on the screen as we see all 0 0, but here one is there and if we go little bit also top also as we going here may be that sometimes it is there. So, these are the, this is not the right output actually we can say the erroneous output and it may produce 0 time most of the time, but sometimes it can produce the value 1 because of the data race.

(Refer Slide Time: 30:11)



```
1  /* The following Java application shows how the transactions in a bank can be carried out concurrently. */
2  @class Account {
3      public int balance;
4      public int accountNo;
5      void displayBalance() {
6          System.out.println("Account No: " + accountNo + "Balance: " + balance);
7      }
8
9      synchronized void deposit(int amount){
10         balance = balance + amount;
11         System.out.println( amount + " is deposited");
12         displayBalance();
13     }
14
15     synchronized void withdraw(int amount){
16         balance = balance - amount;
17         System.out.println( amount + " is withdrawn");
18         displayBalance();
19     }
20 }
21
22 @class TransactionDeposit implements Runnable{
23     int amount;
24     Account accountX;
25     TransactionDeposit(Account x, int amount){
26         accountX = x;
27         this.amount = amount;
28         new Thread(this).start();
29     }
30
31     public void run(){
32         accountX.deposit(amount);
33     }
34 }
35
```

Now, data race because this is the program those are the thread under execution, they are not synchronization under synchronization means whenever one thread is in execution then other threads should not be executed simultaneously having accessing the same variable here, we have made the static variable basically the different thread takes the same value on their own execution that is why the data race is coming. Now this is a one example which can better explain in our real life situation the data race condition.

Now this program, we let us have the discussion on this program then I will explain it is execution and then the inherent what is called the point in it. As we see here we first create a account, this account has 3 fields balance and account number 2 with 2 fields are there and one method display balance. So, a very simple class declaration as we declared here. In addition to this these method has one method is called deposit. So, it basically deposit certain amount here into this account.

So; that means, whenever this method will call the balance whatever the current balance will be increased by the value of amount. So, this is the code it is there and finally, it will print the display balance which will declared here. Similarly there is another method that we have declare here withdraw and it basically once amount will be requested for withdraw will be deducted from the balance and after the withdraw is over it will basically display the balance information.

So, these basically a simple one class method this is just regarding an account holder information; that means, the account holder the name or name is not there account number is here balance is there and the balance information balance status and if the account holder wants to do a deposit. So, he will just call this method or he if we want to withdraw something, you will call this method and this method will be executed there.

Now, here is a question point is that if suppose two customers having the same account number join account number may be they want to issue the two method deposit withdraw or deposit again or withdraw or whatever it is there from the two different terminals then they will be executed from the server point of view. So, it is basically executed as a thread. Now again, just like a data race if it is not properly synchronized then these may leads to the erroneous result say balance is 1000 deposit 500 and withdraw is 300 the result may be sometimes 1300 or result may be sometime 700 so, is not total correct.

Now here is basically the remedy, remedy is that if we want to make the two method synchronized by force then we can create the synchronized q r. So, once the synchronized q r is placed then this deposit and the withdraw will be control in a synchronization that mean only one operation either deposit or withdraw will be executed not the two operations can be executed in concurrent in parallely.

Now having these are the class information. Now here is basically two methods, we declare as a transaction deposit basically in order to access the account having the transaction here. So, integer amount this is the account that mean which account the user wants to access and this is basically the code for this one and finally, these transaction deposit will be executed in parallel. So thread so these basically creating a thread for accessing this deposit method.

(Refer Slide Time: 34:05)

```
D:\Demonstration\Demonstration-XI>javac Demonstration_119.java
D:\Demonstration\Demonstration-XI>java Demonstration_119
900 is deposited
Account No:111Balance: 1500
900 is withdrawn
Account No:111Balance: 600
D:\Demonstration\Demonstration-XI>
```

```
28     new Thread(this).start();
29   }
30   public void run(){
31     accountX.deposit(amount);
32   }
33 }
34 }
35
36 class TransactionWithdraw implements Runnable{
37   int amount;
38   Account accountY;
39
40   TransactionWithdraw(Account y, int amount) {
41     accountY = y;
42     this.amount = amount;
43     new Thread(this).start();
44   }
45
46   public void run(){
47     accountY.withdraw(amount);
48   }
49 }
50
51 class Demonstration_119{
52   public static void main(String args[]) {
53     Account ABC = new Account();
54     ABC.balance = 1000;
55     ABC.accountNo = 111;
56     TransactionDeposit t1;
57     TransactionWithdraw t2;
58     t1 = new TransactionDeposit(ABC, 500);
59     t2 = new TransactionWithdraw(ABC, 900);
60   }
61 }
62 }
```

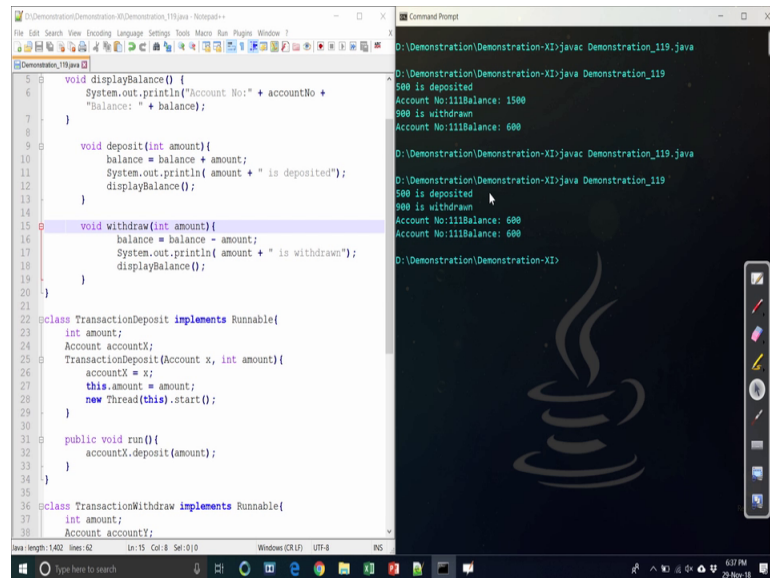
Now, similarly now, similarly there is a transaction withdraw in the same structure as we see here it basically create a thread in this case and then this thread will be again can run concurrently with the transaction deposit. So, these are the two threads in this program. So now, after the two threads are there, this is the main thread this main thread is basically execute the 2 threads that we have created here as we see we have created an account information that mean an account is created this is the account say name of the account B A B balance is 1000 account number is this one and here actually you see t 1 and t 2 the 2 threads are created.

In this case from the same program, but whenever the distributed (Refer Time: 34:50) is there these 2 thread can be executed from the different client and it can pass through the server you can understand that this is basically the request to the server from the 2 client that mean execution of 2 thread t 1 and t 2.

Now if these 2 threads are to be executed by the server in concurrent then definitely data race may occur, but here these thread are created or called for with depositing 500 and withdrawing 9 900 and then what will happen it will give always the correct result. So, in this case 1000 is the balance and after we depositing 500. So, total 1500 then after withdrawing 900, it basically 600. So, this is the output as we see as we see here. So, 500 is deposited this one 900 withdrawn 600. So, this is the correct result.

Now, I can run the same program, but removing the synchronizations it will run again just quickly we just, but in some situation may not always it can give erroneous result.

(Refer Slide Time: 35:55)



The screenshot shows an IDE window on the left with Java code and a Command Prompt window on the right showing the output of the code. The code defines a bank account system with deposit and withdraw methods, and two classes that implement Runnable to perform these actions in parallel threads. The output shows that the deposit and withdraw operations are not synchronized, leading to an incorrect final balance.

```
5 void displayBalance() {
6     System.out.println("Account No: " + accountNo +
7         "Balance: " + balance);
8 }
9
10 void deposit(int amount){
11     balance = balance + amount;
12     System.out.println(amount + " is deposited");
13     displayBalance();
14 }
15
16 void withdraw(int amount){
17     balance = balance - amount;
18     System.out.println(amount + " is withdrawn");
19     displayBalance();
20 }
21
22 class TransactionDeposit implements Runnable{
23     int amount;
24     Account accountX;
25     TransactionDeposit(Account x, int amount){
26         accountX = x;
27         this.amount = amount;
28         new Thread(this).start();
29     }
30
31     public void run(){
32         accountX.deposit(amount);
33     }
34 }
35
36 class TransactionWithdraw implements Runnable{
37     int amount;
38     Account accountY;
```

```
D:\Demonstration\Demonstration> javac Demonstration_119.java
D:\Demonstration\Demonstration> java Demonstration_119
500 is deposited
Account No:111Balance: 1500
500 is withdrawn
Account No:111Balance: 600
D:\Demonstration\Demonstration> java Demonstration_119.java
500 is deposited
500 is withdrawn
Account No:111Balance: 600
Account No:111Balance: 600
D:\Demonstration\Demonstration>
```

Now here, we can see yeah so, this is the right output as we see here just bigger. Now see in the first execution the output was correct, but whereas, these execution the output was not correct in that sense because, they are not executed in synchronization if the amount is different you can get the another result also like this one anyway. So, these basically shows that how the synchronization can help us to synchronize the program ok.

So, we have learned all this topics here and go to the yes fine, but the thing is that this is just only tip of the iceberg, we said say there are many more things that you can consider while you have to practice it more programs are there from the website link, which you have mentioned very beginning, you can follow many more programs, you can access from there and you can run those things there, if you have any doubt any query you are feel free to ask us.

Thank you very much.