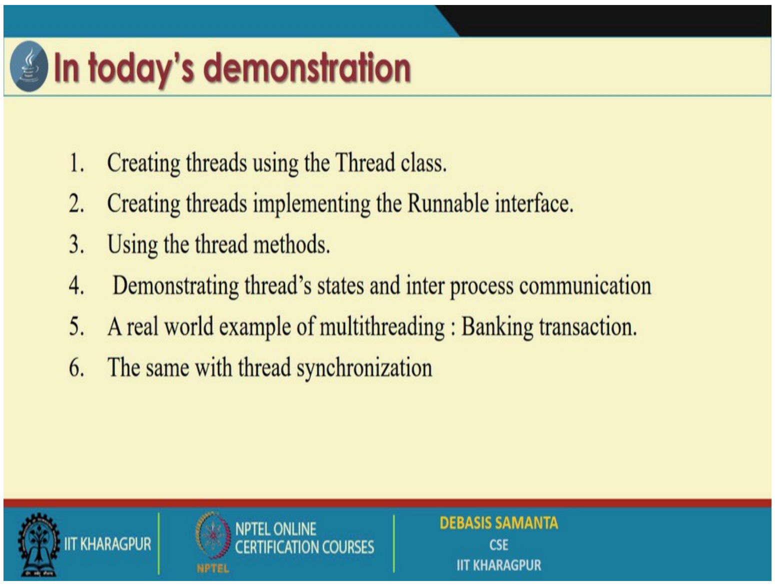
Programming in Java

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

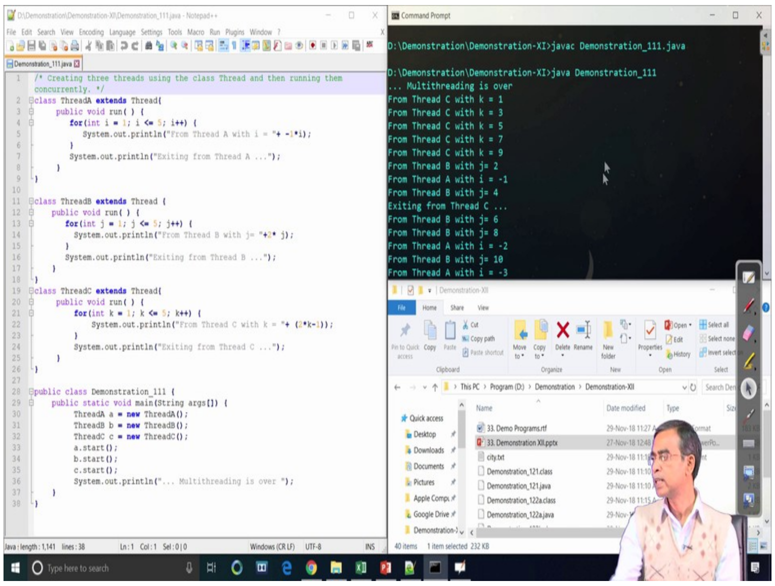
So, in today's demo we will learn everything from the practical point of view, we will see exactly how ThreadCan be created and then those ThreadCan be executed.

And then we have already discussed that there are 2 ways of creating thread. So, using ThreadClassAnd 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 ThreadCan be controlled.

Now finally, we will discussed 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 ThreadClass.

(Refer Slide Time: 02:15)



```java
/* Creating three threads using the class Thread and then running them
concurrently. */
class ThreadA extends Thread{
     public void run( ) {
        for(int i = 1; i <= 5; i++) {
           System.out.println("From Thread A with i = "+ -1*i);
        }
        System.out.println("Exiting from Thread A ...");
     }
}

class ThreadB extends Thread {
    public void run( ) {
       for(int j = 1; j <= 5; j++) {
         System.out.println("From Thread B with j= "+2* j);
       }
       System.out.println("Exiting from Thread B ...");
    }
}
class ThreadC extends Thread{
     public void run( ) {
        for(int k = 1; k <= 5; k++) {
           System.out.println("From Thread C with k = "+ (2*k-1));
        }
        System.out.println("Exiting from Thread C ...");
     }
}

public class Demonstration_111 {
    public static void main(String args[]) {
         ThreadA a = new ThreadA();
         ThreadB b = new ThreadB();
         ThreadC c = new ThreadC();
         a.start();
         b.start();
         c.start();
         System.out.println("... Multithreading is over ");
    }
}
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_111.java

D:\Demonstration\Demonstration-XI>java Demonstration_111
... Multithreading is over
From Thread C with k = 1
From Thread C with k = 3
From Thread C with k = 5
From Thread C with k = 7
From Thread C with k = 9
From Thread B with j= 2
From Thread A with i = -1
From Thread B with j= 4
Exiting from Thread C ...
From Thread B with j= 6
From Thread B with j= 8
From Thread A with i = -2
From Thread B with j= 10
From Thread A with i = -3
```

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 ThreadA and this is basically extends Thread, this means this basically threads using the class Thread 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 ThreadB again it is extend the Thread ClassAnd 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, ThreadC 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.

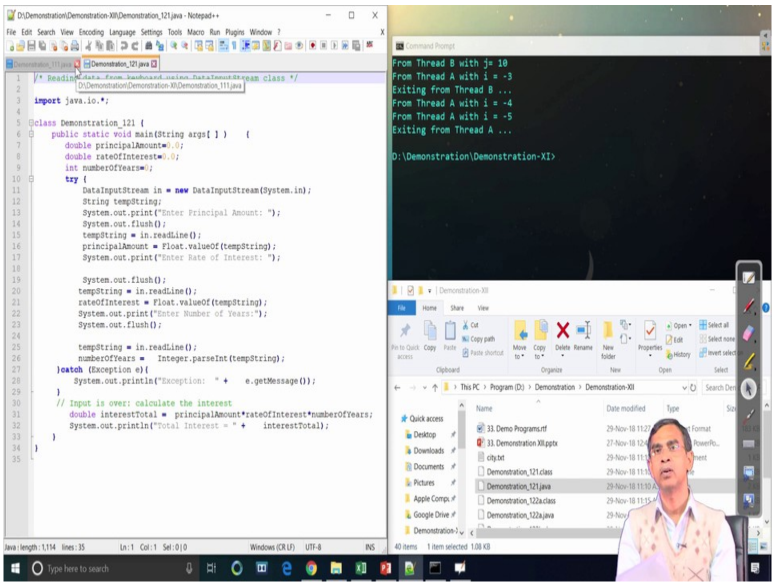
Now so these are the 3 threads. So, our plan is to run these 3 threads simultaneously now here is the main method and in this main method, we see how we can create the 3 3 thread objects A, B and C these are the 3 thread objects for the 3 classes like ThreadA, ThreadB and ThreadC. Now this is the syntax 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.start() this means that ThreadA will run b.start() that this will run c.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 ThreadA will run this means this thread will print the negative numbers ThreadB when it will run it will print odd even numbers ThreadC 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_111 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 ThreadC ThreadB A B B they are basically in random order somehow ThreadC 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)

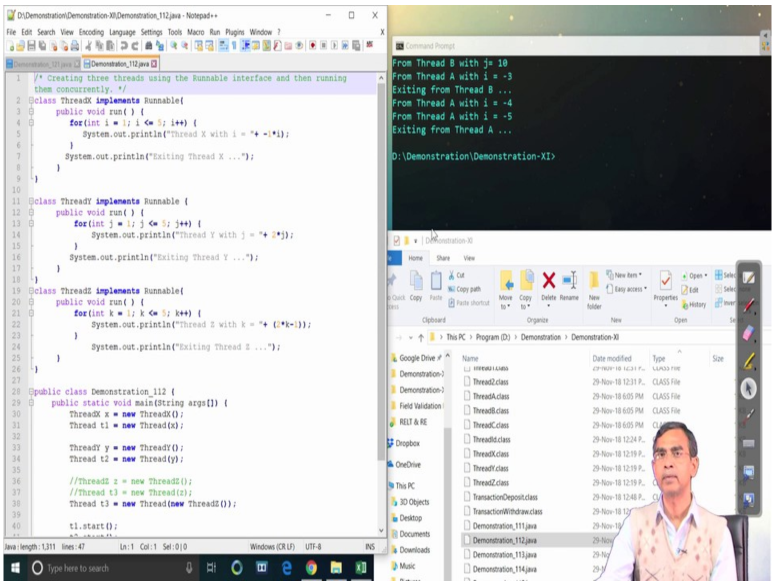


```java
/* Reading data from keyboard using DataInputStream class */

import java.io.*;

class Demonstration_121 {
    public static void main(String args[ ] )    {
        double principalAmount=0.0;
        double rateOfInterest=0.0;
        int numberOfYears=0;
        try {
            DataInputStream in = new DataInputStream(System.in);
            String tempString;
            System.out.print("Enter Principal Amount: ");
            System.out.flush();
            tempString = in.readLine();
            principalAmount = Float.valueOf(tempString);
            System.out.print("Enter Rate of Interest: ");

            System.out.flush();
           tempString = in.readLine();
           rateOfInterest = Float.valueOf(tempString);
           System.out.print("Enter Number of Years:");
           System.out.flush();

           tempString = in.readLine();
           numberOfYears =   Integer.parseInt(tempString);
       }catch (Exception e){
          System.out.println("Exception:  " +    e.getMessage());
       }
       // Input is over: calculate the interest
          double interestTotal =  principalAmount*rateOfInterest*numberOfYears;
          System.out.println("Total Interest = " +    interestTotal);
     }
}
```

```
From Thread B with j= 10
From Thread A with i = -3
Exiting from Thread B ...
From Thread A with i = -4
From Thread A with i = -5
Exiting from Thread A ...

D:\Demonstration\Demonstration-XI>
```

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 112 ok.

(Refer Slide Time: 07:09)



```java
/* Creating three threads using the Runnable interface and then running
them concurrently. */
class ThreadX implements Runnable{
     public void run( ) {
        for(int i = 1; i <= 5; i++) {
           System.out.println("Thread X with i = "+ -1*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 ...");
     }
}

public class Demonstration_112 {
    public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);

        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);

        //ThreadZ z = new ThreadZ();
        //Thread t3 = new Thread(z);
        Thread t3 = new Thread(new ThreadZ());

        t1.start();
```

```
From Thread B with j= 10
From Thread A with i = -3
Exiting from Thread B ...
From Thread A with i = -4
From Thread A with i = -5
Exiting from Thread A ...

D:\Demonstration\Demonstration-XI>
```

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 ThreadX 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 ThreadA extends Thread class, in the last example similarly ThreadY is the same version of ThreadB and this is the implementation of run method for this ThreadX and similarly this is the implementation of run method for ThreadZ, it basically implementing Runnable interface.

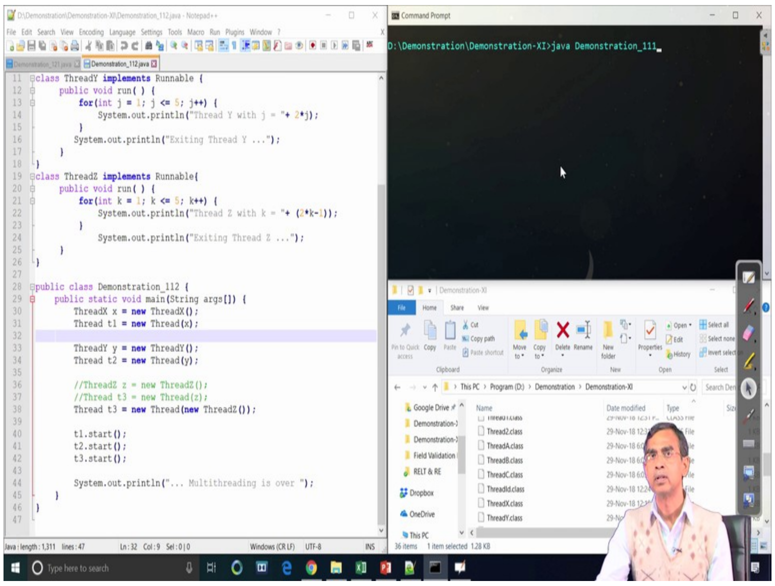
So, as we see there are 3 threads we have created using Runnable interface. Now let us see how we can create the. So, these are the basically class declaration for the 3 threads and now this is a main method, this method is basically creates the 3 threads as we defined ThreadX ThreadY and Thread Z. Now 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 t1 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 t1 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 t1 for the ThreadX likewise, these basically t2 is the thread 2 for the class object y and thread ThreadY actually and this is the another way short cut method also we can use.

So, t3 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 t1, t2 and t3 and now we are in a position to run them.

(Refer Slide Time: 10:03)

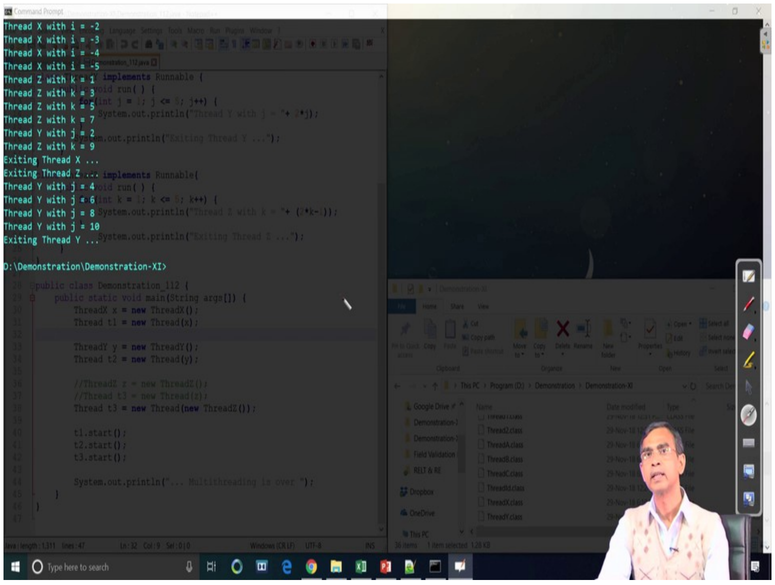


```java
class ThreadY implements Runnable {
    public void run( ) {
        for(int j = 1; j <= 5; j++) {
            System.out.println("Thread Y with j = "+ 2*j);
        }
        System.out.println("Exiting Thread Y ...");
    }
}
class ThreadZ implements Runnable{
    public void run( ) {
    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

public class Demonstration_112 {
   public static void main(String args[]) {
        ThreadX x = new ThreadX();
        Thread t1 = new Thread(x);

        ThreadY y = new ThreadY();
        Thread t2 = new Thread(y);

        //ThreadZ z = new ThreadZ();
        //Thread t3 = new Thread(z);
        Thread t3 = new Thread(new ThreadZ());

        t1.start();
        t2.start();
        t3.start();

        System.out.println("... Multithreading is over ");
   }
}
```

```
D:\Demonstration\Demonstration-XI>java Demonstration_111_
```

So here again, we run t1.start(), t2.start(), t3.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.

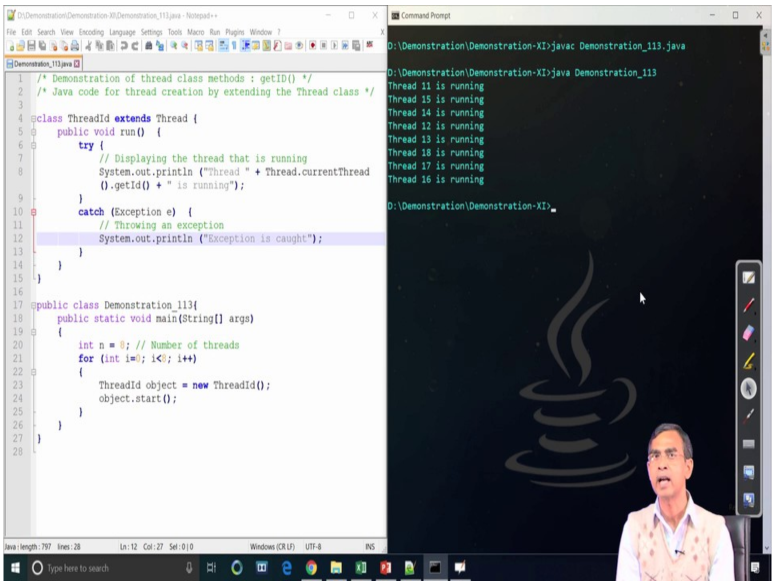
(Refer Slide Time: 10:39)



```
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 Z with k = 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 ...

D:\Demonstration\Demonstration-XI>
```

So, this ThreadA s 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 parallely, 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 interleaved 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)



```java
/* Demonstration of thread class methods : getID() */
/* Java code for thread creation by extending the Thread class */

class ThreadId extends Thread {
    public void run()  {
        try {
            // Displaying the thread that is running
            System.out.println ("Thread " + Thread.currentThread
            ().getId() + " is running");
        }
        catch (Exception e)  {
            // Throwing an exception
            System.out.println ("Exception is caught");
        }
    }
}

public class Demonstration_113{
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i=0; i<8; i++)
        {
            ThreadId object = new ThreadId();
            object.start();
        }
    }
}
```

```
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>_
```

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 Id of a Thread can be accessed by a method called the getId, which is defined in the class Thread in the Java.lang.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 ThreadId threadId 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,.

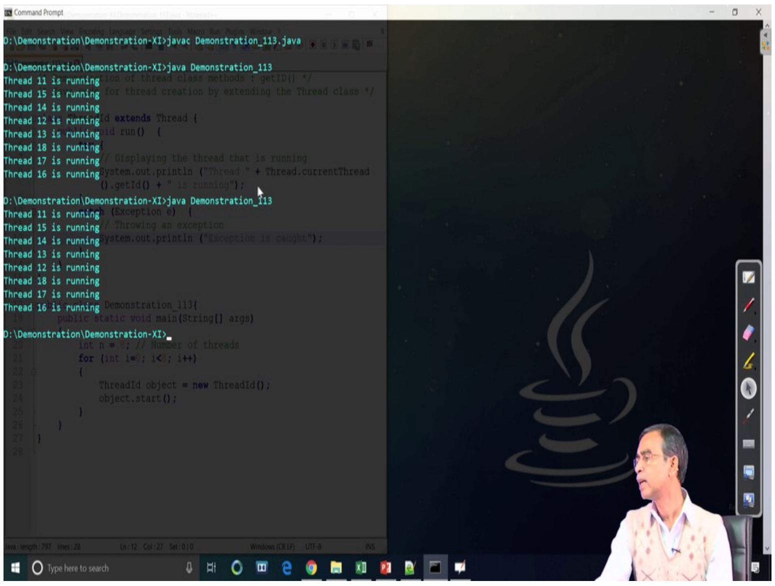
Now here, the run method basically thread has it is own abstract run method, but whenever you have to extend it we have to basically implement override this run method.

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 getId means for that thread the Id 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 Id 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 threads will be executed we are basically running for the same class ThreadId, 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 Id.

(Refer Slide Time: 14:07)

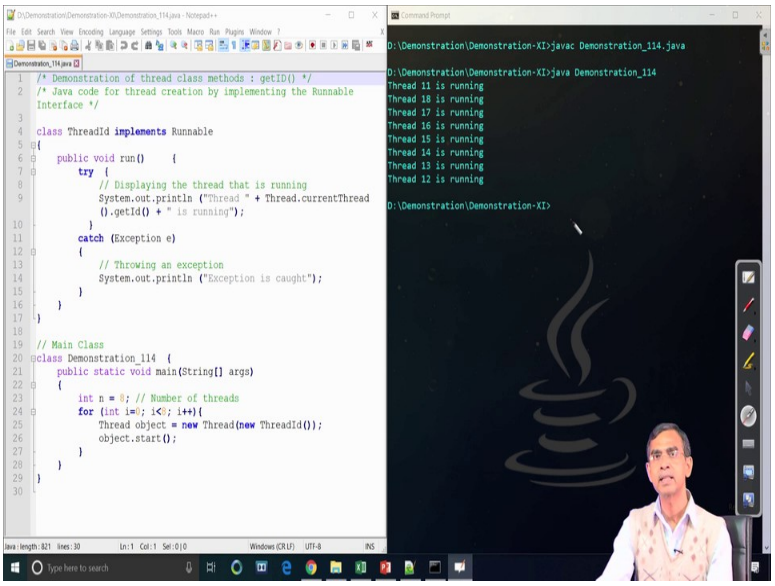


```
D:\Demonstration\Demonstration-XI>javac Demonstration_113.java

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>
```

So, as we see here thread that is I1 is running thread I5 is running I4 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, Id 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)



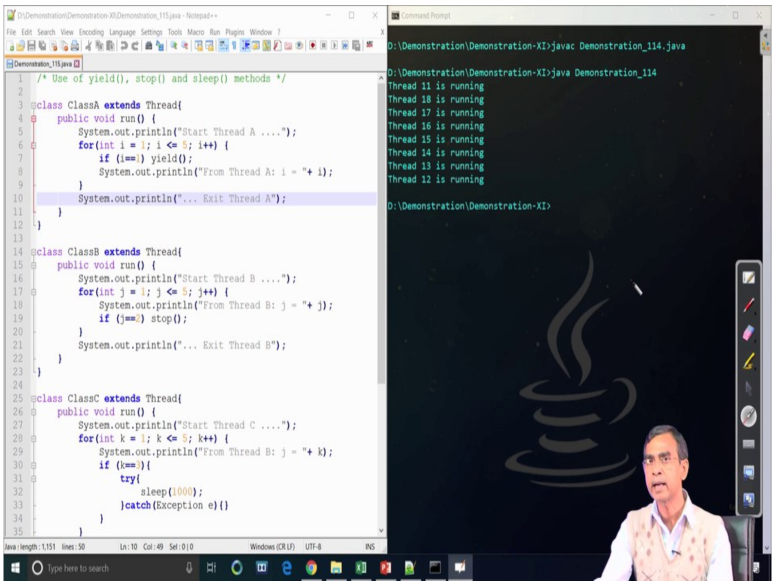
```java
/* Demonstration of thread class methods : getID() */
/* Java code for thread creation by implementing the Runnable
Interface */

class ThreadId implements Runnable
{
    public void run()     {
        try  {
            // Displaying the thread that is running
            System.out.println ("Thread " + Thread.currentThread
            ().getId() + " is running");
        }
        catch (Exception e)
        {
            // Throwing an exception
            System.out.println ("Exception is caught");
        }
    }
}

// Main Class
class Demonstration_114  {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i=0; i<8; i++){
            Thread object = new Thread(new ThreadId());
            object.start();
        }
    }
}
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_114.java

D:\Demonstration\Demonstration-XI>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-XI>
```

As we have seen in the last program, we have created the Thread class, you have we have extended the Thread ClassAgain 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)



```java
/* Use of yield(), stop() and sleep() methods */

class ClassA extends Thread{
    public void run() {
        System.out.println("Start Thread A ....");
        for(int i = 1; i <= 5; i++) {
            if (i==1) yield();
            System.out.println("From Thread A: i = "+ i);
        }
        System.out.println("... Exit Thread A");
    }
}

class ClassB extends Thread{
    public void run() {
        System.out.println("Start Thread B ....");
        for(int j = 1; j <= 5; j++) {
            System.out.println("From Thread B: j = "+ j);
            if (j==2) stop();
        }
        System.out.println("... Exit Thread B");
    }
}

class ClassC extends Thread{
    public void run() {
        System.out.println("Start Thread C ....");
        for(int k = 1; k <= 5; k++) {
            System.out.println("From Thread B: j = "+ k);
            if (k==3){
                try{
                    sleep(1000);
                }catch(Exception e){}
            }
        }
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_114.java

D:\Demonstration\Demonstration-XI>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-XI>
```

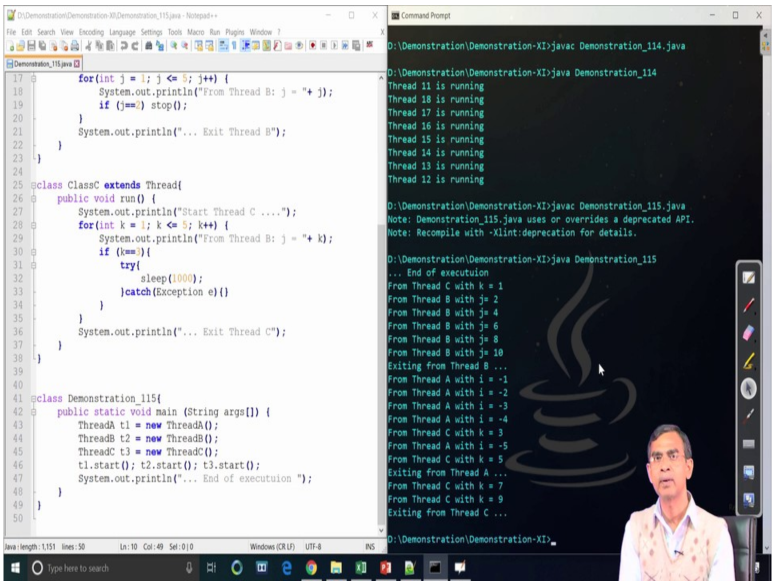
For example, there is yield method, stop method, suspend method, resume method, sleep method, whatever it is there.

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 ClassA 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 ThreadA 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 ThreadA 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 ClassB extends Thread s 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 ClassC 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)



```
        for(int j = 1; j <= 5; j++) {
            System.out.println("From Thread B: j = "+ j);
            if (j==2) stop();
        }
        System.out.println("... Exit Thread B");
    }
}

class ClassC extends Thread{
    public void run() {
        System.out.println("Start Thread C ....");
        for(int k = 1; k <= 5; k++) {
            System.out.println("From Thread B: j = "+ k);
            if (k==3){
                try{
                    sleep(1000);
                }catch(Exception e){}
            }
        }
        System.out.println("... Exit Thread C");
    }
}


class Demonstration_115{
    public static void main (String args[]) {
        ThreadA t1 = new ThreadA();
        ThreadB t2 = new ThreadB();
        ThreadC t3 = new ThreadC();
        t1.start(); t2.start(); t3.start();
        System.out.println("... End of executuion ");
    }
}
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_114.java

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

D:\Demonstration\Demonstration-XI>java Demonstration_115
... End of executuion
From Thread C with k = 1
From Thread B with j= 2
From Thread B with j= 4
From Thread B with j= 6
From Thread B with j= 8
From Thread B with j= 10
Exiting from Thread B ...
From Thread A with i = -1
From Thread A with i = -2
From Thread A with i = -3
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 ...

D:\Demonstration\Demonstration-XI>_
```

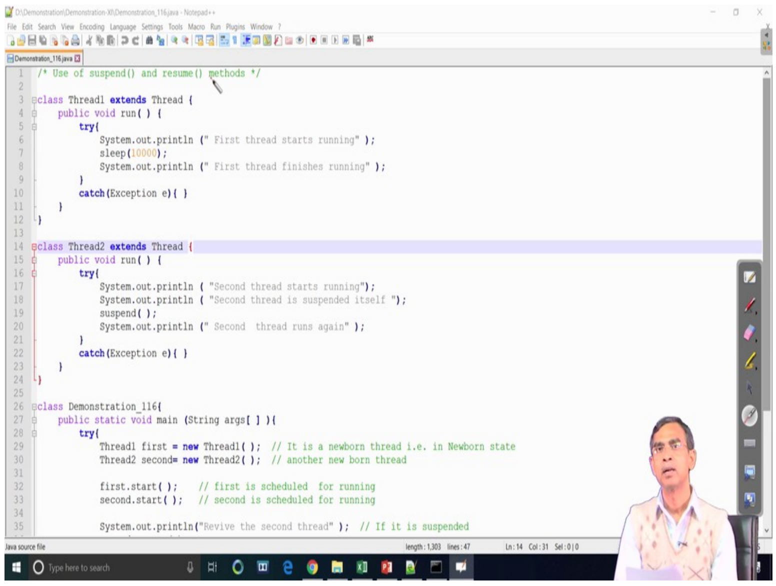
So, these are the 3 Thread classes we have created having their own mechanism and now we in the main method, we can create the 3 objects of this 3 Thread classes t 1, t 2, t 3 and then we just start their execution here and then finally, one the thread is finished it basically print that end of the thread execution, but as it is here in this main method actually in parallel 4 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.

So, as we see the output seen 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 ThreadC, ThreadB, ThreadB in a random fashion as we see the different threads executed and also we can see the second thread, ThreadB 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)



```
/* Use of suspend() and resume() methods */

class Thread1 extends Thread {
    public void run( ) {
        try{
            System.out.println (" First thread starts running" );
            sleep(10000);
            System.out.println (" First thread finishes running" );
        }
        catch(Exception e){ }
    }
}

class Thread2 extends Thread {
    public void run( ) {
        try{
            System.out.println ( "Second thread starts running");
            System.out.println ( "Second thread is suspended itself ");
            suspend( );
            System.out.println (" Second  thread runs again" );
        }
        catch(Exception e){ }
    }
}

class Demonstration_116{
    public static void main (String args[ ] ){
        try{
            Thread1 first = new Thread1( );  // It is a newborn thread i.e. in Newborn state
            Thread2 second= new Thread2( );  // another new born thread

            first.start( );    // first is scheduled  for running
            second.start( );   // second is scheduled for running

            System.out.println("Revive the second thread" );  // If it is suspended
```

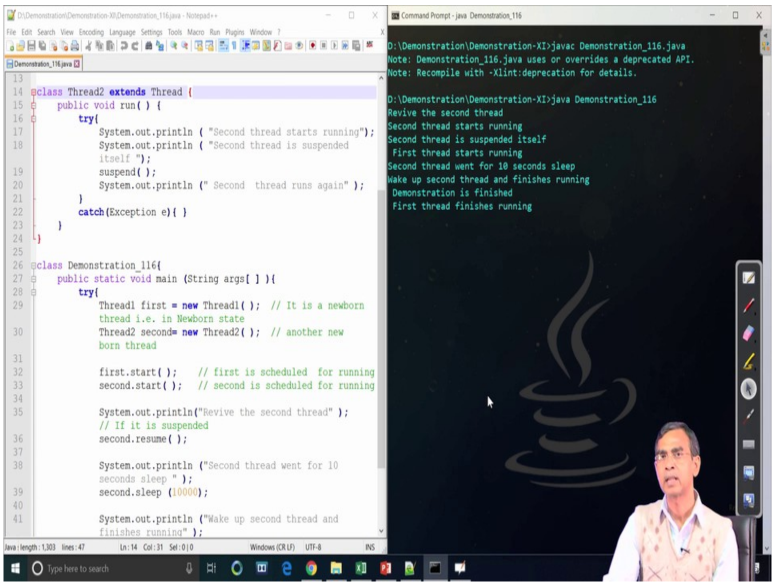
But, if you run in parallel it may take 5 seconds both the program will run, but it will the maximally utilize the CPU resources other resources, those are required in your program execution can be maximally utilized. So, this is the purpose of the thesis purpose of the threads execution.

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 Thread1, 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 Thread2 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 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 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)



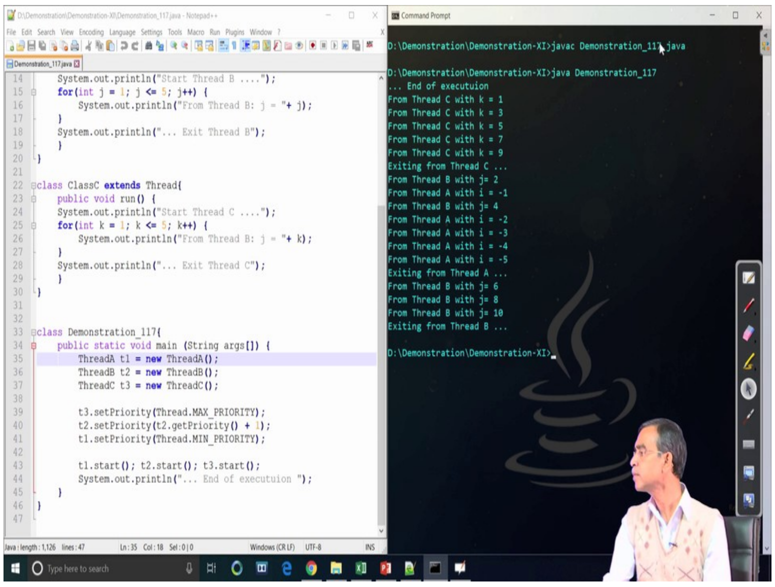
```java
class Thread2 extends Thread {
    public void run( ) {
        try{
            System.out.println ( "Second thread starts running");
            System.out.println ( "Second thread is suspended
            itself ");
            suspend( );
            System.out.println (" Second  thread runs again" );
        }
        catch(Exception e){ }
    }
}

class Demonstration_116{
    public static void main (String args[ ] ){
        try{
            Thread1 first = new Thread1( );  // It is a newborn
            thread i.e. in Newborn state
            Thread2 second= new Thread2( );  // another new
            born thread

            first.start( );    // first is scheduled  for running
            second.start( );   // second is scheduled for running

            System.out.println("Revive the second thread" );
            // If it is suspended
            second.resume( );

            System.out.println ("Second thread went for 10
            seconds sleep " );
            second.sleep (10000);

            System.out.println ("Wake up second thread and
            finishes running" );
```

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

D:\Demonstration\Demonstration-XI>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)



```
        System.out.println("Start Thread B ....");
        for(int j = 1; j <= 5; j++) {
            System.out.println("From Thread B: j = "+ j);
        }
        System.out.println("... Exit Thread B");
        }
}

class ClassC extends Thread{
        public void run() {
        System.out.println("Start Thread C ....");
        for(int k = 1; k <= 5; k++) {
            System.out.println("From Thread B: j = "+ k);
        }
        System.out.println("... Exit Thread C");
        }
}


class Demonstration_117{
        public static void main (String args[]) {
            ThreadA t1 = new ThreadA();
            ThreadB t2 = new ThreadB();
            ThreadC t3 = new ThreadC();

            t3.setPriority(Thread.MAX_PRIORITY);
            t2.setPriority(t2.getPriority() + 1);
            t1.setPriority(Thread.MIN_PRIORITY);

            t1.start(); t2.start(); t3.start();
            System.out.println("... End of executuion ");
        }
}
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_117.java

D:\Demonstration\Demonstration-XI>java Demonstration_117
... End of executuion
From Thread C with k = 1
From Thread C with k = 3
From Thread C with k = 5
From Thread C with k = 7
From Thread C with k = 9
Exiting from Thread C ...
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 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= 6
From Thread B with j= 8
From Thread B with j= 10
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.

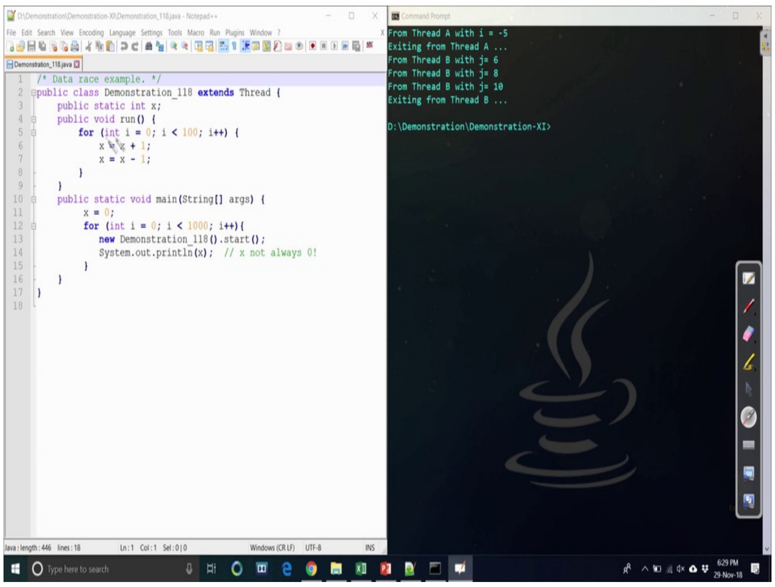
Now, the thread we can assign priority values to all these threads here now, this program will explain how we can create 3 threads and among these 3 threads we can assign the priority. So, ClassA is a 1 thread, this thread has a simple loop i equals to 1 to 5 and then this is the simple loop execution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

Similarly ClassB 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 threads t1, t 2, t 2 we create 3 threads t1, t2 and t3 here for the class ThreadA ThreadB and ThreadC and then here the method set priority which is defined in the class Thread and then we plus argument as Thread.MAX_PRIORITY, this value is already defined in the ThreadClass.

So here, basically this basically t 3 has been assigned the maximum priority and as we see t1 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 t3, 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, t3 because t3 the ThreadC 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)



```java
/* Data race example. */
public class Demonstration_118 extends Thread {
    public static int x;
    public void run() {
        for (int i = 0; i < 100; i++) {
            x = x + 1;
            x = x - 1;
        }
    }
    public static void main(String[] args) {
        x = 0;
        for (int i = 0; i < 1000; i++){
            new Demonstration_118().start();
            System.out.println(x);  // x not always 0!
        }
    }
}
```

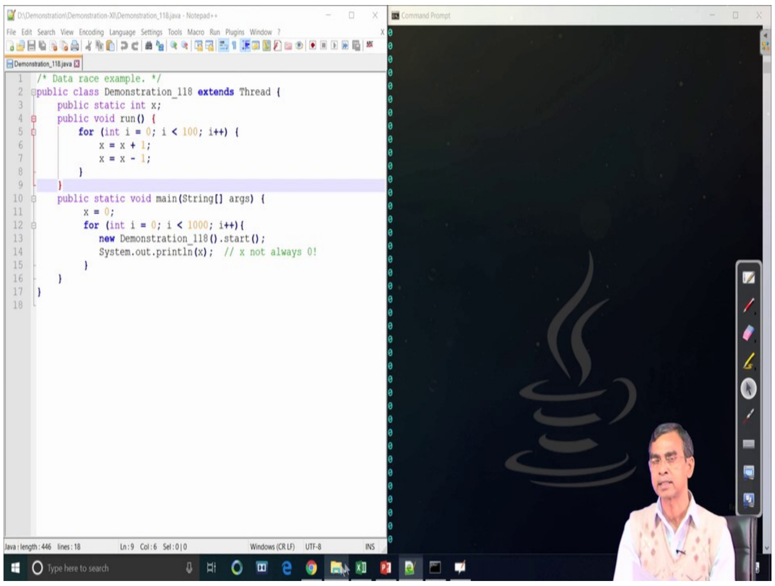
```
From Thread A with i = -5
Exiting from Thread A ...
From Thread B with j= 6
From Thread B with j= 8
From Thread B with j= 10
Exiting from Thread B ...

D:\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= x +1 , x =x-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 produce, 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.

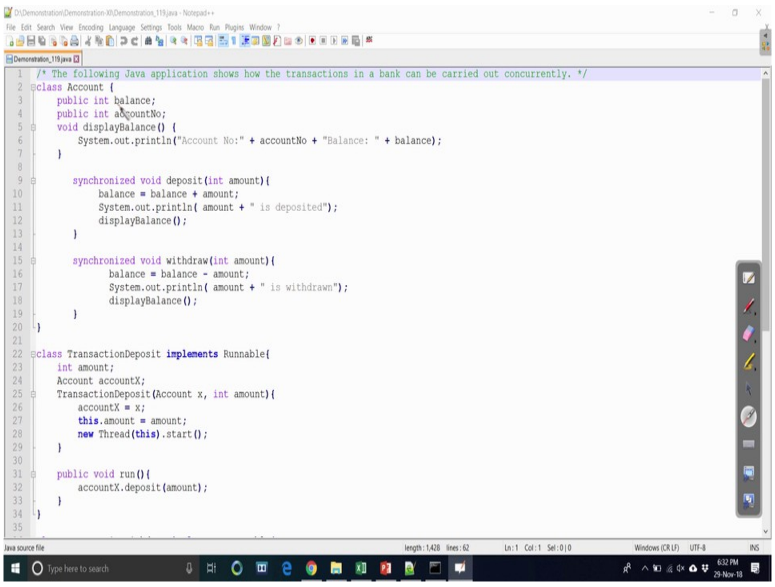
(Refer Slide Time: 29:37)



```java
/* Data race example. */
public class Demonstration_118 extends Thread {
    public static int x;
    public void run() {
        for (int i = 0; i < 100; i++) {
            x = x + 1;
            x = x - 1;
        }
    }
    public static void main(String[] args) {
         x = 0;
         for (int i = 0; i < 1000; i++){
            new Demonstration_118().start();
            System.out.println(x);  // x 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)



```java
/* The following Java application shows how the transactions in a bank can be carried out concurrently. */
class Account {
    public int balance;
    public int accountNo;
    void displayBalance() {
        System.out.println("Account No:" + accountNo + "Balance: " + balance);
    }

        synchronized void deposit(int amount){
            balance = balance + amount;
            System.out.println( amount + " is deposited");
            displayBalance();
        }

        synchronized void withdraw(int amount){
             balance = balance - amount;
             System.out.println( amount + " is withdrawn");
             displayBalance();
        }
}

class TransactionDeposit implements Runnable{
    int amount;
    Account accountX;
    TransactionDeposit(Account x, int amount){
        accountX = x;
        this.amount = amount;
        new Thread(this).start();
    }

    public void run(){
        accountX.deposit(amount);
    }
}
```

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 in 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 (accountNo) with 2 fields are there and one method displayBalance. 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.

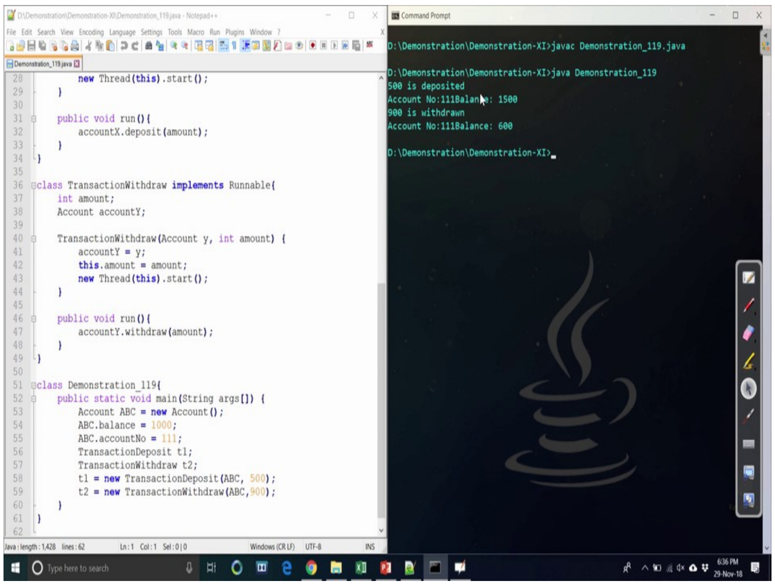
So, these basically a simple one class method this is just regarding an account holder information; that means, the account holder the name or name is not there account number is here balance is there and the balance information balance status and if the account holder wants to do a deposit. So, he will 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 joint 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 keyword. So, once the synchronized keyword 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)



```java
            new Thread(this).start();
        }

        public void run(){
            accountX.deposit(amount);
        }
}

class TransactionWithdraw implements Runnable{
        int amount;
        Account accountY;

        TransactionWithdraw(Account y, int amount) {
            accountY = y;
            this.amount = amount;
            new Thread(this).start();
        }

        public void run(){
            accountY.withdraw(amount);
        }
}

class Demonstration_119{
        public static void main(String args[]) {
            Account ABC = new Account();
            ABC.balance = 1000;
            ABC.accountNo = 111;
            TransactionDeposit t1;
            TransactionWithdraw t2;
            t1 = new TransactionDeposit(ABC, 500);
            t2 = new TransactionWithdraw(ABC,900);
        }
}
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_119.java

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

D:\Demonstration\Demonstration-XI>
```

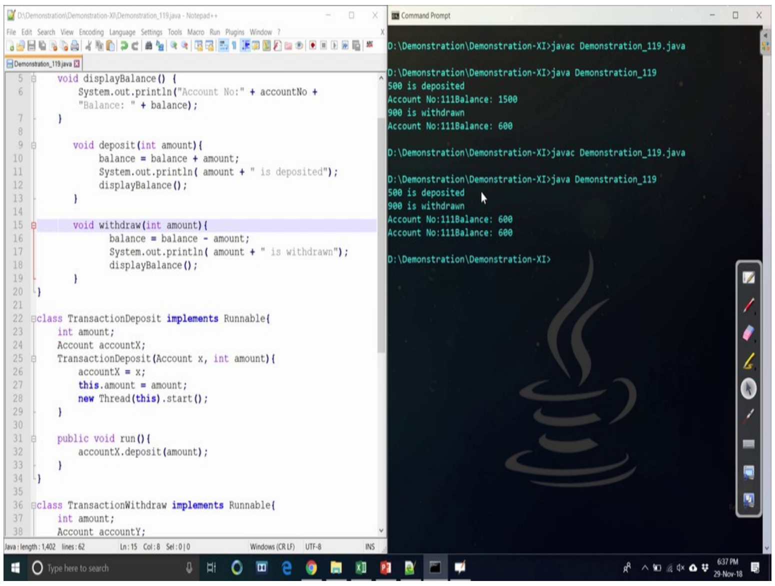
Now, similarly now, similarly there is a transaction withdraw in the same structure as we see here it basically create a thread in this case and then this thread will be again can run concurrently with the transaction deposit. So, these are the two threads in this program. So now, after the two threads 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)



```java
    void displayBalance() {
        System.out.println("Account No:" + accountNo +
        "Balance: " + balance);
    }

       void deposit(int amount){
            balance = balance + amount;
            System.out.println( amount + " is deposited");
            displayBalance();
       }

       void withdraw(int amount){
              balance = balance - amount;
              System.out.println( amount + " is withdrawn");
              displayBalance();
       }
}

class TransactionDeposit implements Runnable{
    int amount;
    Account accountX;
    TransactionDeposit(Account x, int amount){
        accountX = x;
        this.amount = amount;
        new Thread(this).start();
    }

    public void run(){
        accountX.deposit(amount);
    }
}

class TransactionWithdraw implements Runnable{
    int amount;
    Account accountY;
```

```
D:\Demonstration\Demonstration-XI>javac Demonstration_119.java

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

D:\Demonstration\Demonstration-XI>javac Demonstration_119.java

D:\Demonstration\Demonstration-XI>java Demonstration_119
500 is deposited
900 is withdrawn
Account No:111Balance: 600
Account No:111Balance: 600

D:\Demonstration\Demonstration-XI>
```

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.