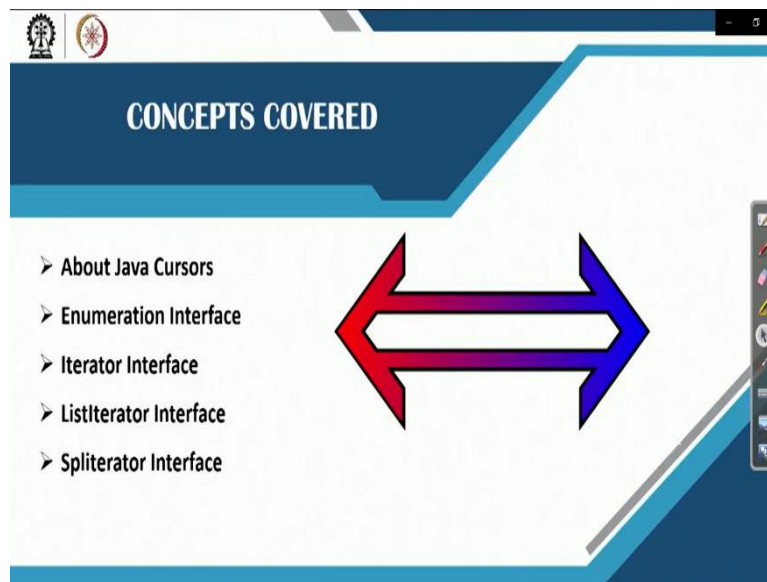


**Data Structures and Algorithms Using Java**  
**Professor Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture - 60**  
**Java Cursors**

In this course, we have covered many different type of collection. The collections may be in the form of an array, or it is a form of a set, or tree, or table, and graph whatever it is there. Now, there is one important aspect, which basically I have used, although I did not discuss in details. That how a collection can be viewed. Now, viewing a collection is basically called as a cursoring. Cursorsing means how to move from one element in a collection to others.

(Refer Slide Time: 01:08)



So regarding these things, java developers have taken special interest to give many ways to move across the different collections. So in this lecture, we will try to cover about java cursors. And there are mainly four different types of cursors are there.

One is called the enumeration interface, the iterator interface, list iterator interface, spliterator interface. In fact different collections, they basically implements all the interface so that the different way a collection can be traverse.

(Refer Slide Time: 01:50)

**About Java Cursors**

NPTEL Online Certification Course  
IIT Kharagpur

**Java cursors**

A Java cursor is a pointer (more precisely loop indicator), which is used to iterate (loop or cycle or visit) or traverse or retrieve collection elements one by one.

When you are dealing with a collection, you have to perform CRUD operations.

The CRUD operations in JCF implies the following:

- Create:** Adding new elements to Collection object.
- Read:** Retrieving elements from Collection object.
- Update:** Updating or setting existing elements in Collection object.
- Delete:** Removing elements from Collection object.

You will see there are many classes in the JCF loaded with many methods in each to accomplish the CRUD operations. In addition to the CRUD operations, it is also an important aspect to traverse or visit each element in the cursor. For this very reason, Java developer introduces a concept called **Java cursor**.

So we will discuss about the different cursors. And the cursors is, basically, as you know, so far this collection is concerned, with collections we have to perform four different operations. One is called the create, create a collection; then read, we have to read the different elements in a collection; update collection if you want to modify; then delete is a removing some elements from the collection.

Now, so there are many classes in the java collection framework, which basically with many methods in order to accomplish all these operations. All these operations are together called the

CRUD operation. So there are, in addition to these CRUD operation, these means related to I mean, read, insert, delete, move, whatever it is, there are certain operation is required that operation is related to traversing or visiting each element in the cursor.

So for this purpose Java developer introduce a concept. This concept is called java cursors concept.

(Refer Slide Time: 03:04)

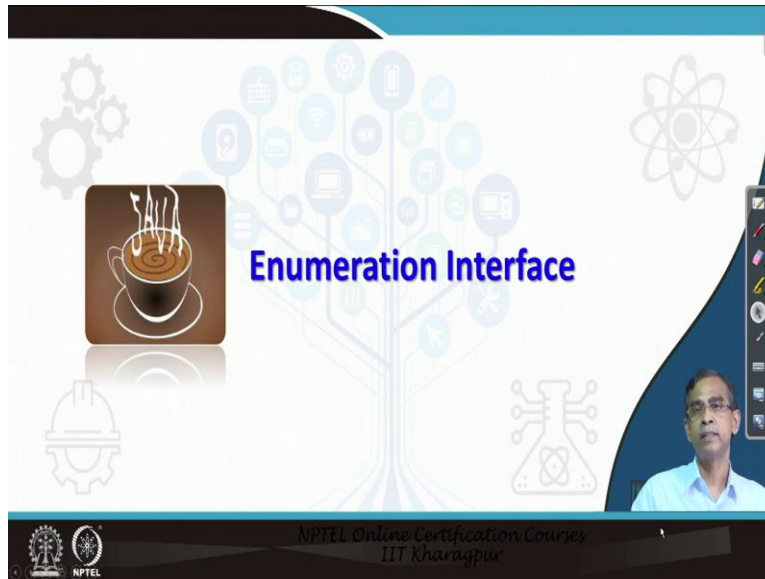
**List of Java cursors**

- Enumeration
- Iterator
- ListIterator
- Spliterator

Let us learn each of the above-mentioned Java cursors, in details with appropriate illustrations.

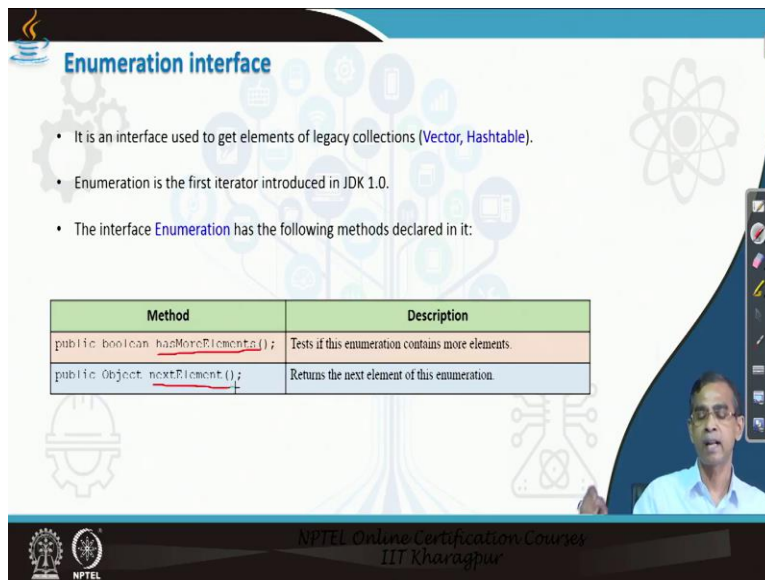
Now, so far, these java cursor are concerned. There are four different type enumeration iterator, list iterator, and then spliterator.

(Refer Slide Time: 03:18)



So let us learn about each iterators, each cursor individually one by one. As the detailed discussion is really not possible, so I should suggest you to the supplementary material that I have provided at the end of this link or the lecture. There is a link you can follow this link to get it.

(Refer Slide Time: 03:34)



Now, let us first come to the discussion of enumeration interface. This interface is basically used to traverse the legacy class, namely vector, and hash table. We have discussed about the java

legacy class and it is there. So enumeration is basically the first iterator or it is basically the cursor introduced at very beginning in the JDK 1.0.

This interface has, this is an interface, as you know, so it does not have any constructor defined in it. But there are methods are there; only two methods those are defined. These two methods are called has more element, it basically check that if the collection has more elements or not, if it has no elements while it is traversing then it return false otherwise return true.

And next element is return the next element that basically currently it is traversing. So these are the two methods by which we can traverse enumeration of type it is there.

(Refer Slide Time: 04:39)

**Example 60.1 : Demonstration of enumeration**

The **Collection** class defined in `java.util` package has its own implementation of the interface **Enumeration**. **Enumerations** are also used to specify the input streams to a **SequenceInputStream**. You can create **Enumeration** object by calling `elements()` method of **Vector** class on any **Vector** object. For example, if `v` denotes an object of the class **Vector** class, then `e` is an object of type **Enumeration** referring to `v` is:

```
Enumeration(e) - v.elements();
```

```
// Java program to demonstrate enumeration
import java.util.Enumeration;
import java.util.Vector;

public class EnumerationTest {
    public static void main(String[] args) {
        // Create a vector and print its contents
        Vector v = new Vector();
        for (int i = 0; i < 10; i++)
            v.add(i);
        System.out.println(v);
    }
} // Continued to next...
```

The slide includes a video inset of a man speaking and the NPTEL logo at the bottom.

And here is an example that you can think about. So let us see this example that we can use it. And this basically demonstrate the enumeration as a cursor. We can create a vector as a collection objects. So let us create a vector as a collection. And we just add some element into the vector. So this basically create the vector objects.

Now, vector object is created. Now, simply by putting this print ln statement for this vector v, it basically gives the overall view or whole view of the vector objects. But we want to parse, we want to scan it or rather traverse it one by one. Then how it can be done?

So this basically done by creating some elements e of type enumerate e and that basically elements method which is define in the vector class can be called. So either v is a vector. If we call the method elements then it, basically the enumerated, which basically is a traverse form of all the elements those are there in the vector v class.

Now, you can recall when we are discussing about the file input-output system, there we can use the sequence input stream, there means we can pass the two or more files and then it basically traverse one file at a time where the file is basically is a collection and then it basically traverse and then concatenate into a another third file that we have already checked it. The sequence input stream class, while we are discussing in the discussion of file IO.

Now, let us continue this program again. We can create one enumerator for the vector sets, vector collection v.

(Refer Slide Time: 06:27)

**Example 60.1 : Demonstration of enumeration**

```
// ... Continued from previous

/*Declare an enumerator to the collection v At the beginning e points to
index just before the first element in v */
Enumeration e = v.elements();

while (e.hasMoreElements()) { // Enumerate each element one-by-one.
    int i = (Integer)e.nextElement(); //Moving cursor to next element
    System.out.print(i + " "); // Print the current element.
}
}
```

NPTEL Online Certification Course  
IIT Kharagpur

And this is the next part of the program that you can check it here. So these basically create the enumerator for all the elements v. Now, this enumerator can be traversed. Here is has more elements that mean we just starting from the very beginning of the elements, which is there, and then we can just parse it.

Here, just as the enumerator are the integer of type, so we need to casting it. So that is why integer is there. So it basically written the i. So next element written, the next element in the enumerator list, it basically casted in integer form to and then finally print it.

So this will basically print all the elements those are there. So system dot out dot print ln v and this basically is basically in this case, give us both same, similar output as you can notice.

(Refer Slide Time: 07:19)

The slide is titled "Limitations of Enumeration" and features a list of four bullet points. The first point states it is applicable to only legacy classes like Vector and HashTable. The second point notes its long method names: hasMoreElements() and nextElement(). The third point highlights that it only supports read operations in CRUD, lacking create, update, and delete. The fourth point mentions it only supports forward iteration, earning it the name 'uni-directional cursor'. A video feed of a speaker is visible in the bottom right corner of the slide area.

- It is applicable to only Collection of legacy classes, like Vector and HashTable.
- Compare to other cursors, it has very lengthy method names: `hasMoreElements()` and `nextElement()`.
- In CRUD operations, it supports only read operation. It does not support create, update and delete operations.
- It supports only forward direction iteration. That's why it is also known as uni-directional cursor.

Next, we will discuss about *Iterator* with some suitable examples.

NPTEL Online Certification Courses  
IIT Kharagpur

So this is the cursor for the enumeration type. And there are few points that you should note that it is applicable to collection of only legacy class. We cannot apply this kind of program to other collection, like array, list or, t list, or hash may like this one.

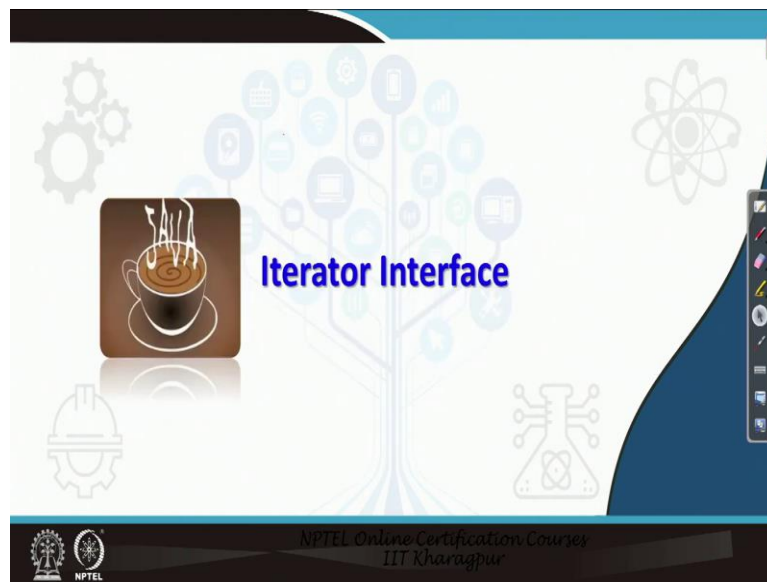
And it is basically criticized that it has very long name, absolute no, so for this, I believe. Because naming is good to understand about it, so this is some people criticize it. And then in CRUD operation, it supports only read operation because it cannot, this is a one important thing it does not support create, update and delete while traversing. That is one important consideration that you should note.

Because while traversing, only printing may not be good one, but while traversing, if you perform certain CRUD operation like creating or some adding some element into it, or deleting, or updating, then it does not support this kind of iterator or traversers. But there are some other



traversers, it is possible that can help it. And moreover, it only supports forward direction traversing. That is how it is called the unidirectional cursor.

(Refer Slide Time: 08:27)



The slide is titled 'Iterator interface' and contains the following text and list:

The limitations in Enumeration interface had been addressed in JDK 1.2, and introduced a better Java cursor called Iterator.

- It is a universal iterator as you can apply it to any Collection object like Set, List, Queue, Deque and also in all implemented classes of Map interface.
- By using Iterator, you can perform both read and remove operations.
- The Iterator interface is fully implemented by Collection classes.
- The Iterator interface defines three methods as listed below.

Method	Description
<code>public boolean hasNext ();</code>	Returns true if the iterator has more elements.
<code>public Object next ();</code>	Returns the next element in the iterator.
<code>public void remove ();</code>	Remove the next element in the iterator. This method can be called only once per call to next().

The NPTEL logo and 'NPTEL Online Certification Courses IIT Kharagpur' are visible at the bottom. A small video inset of a man is visible in the bottom right corner.

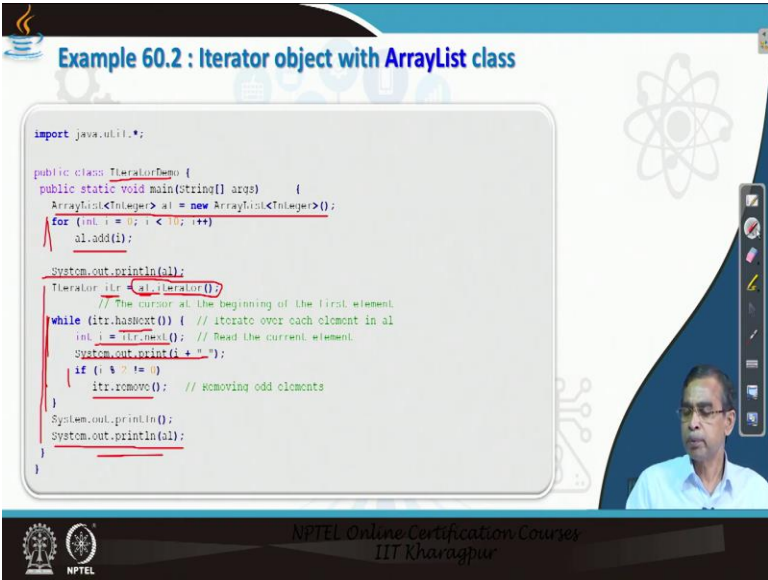
Let us discuss another cursor that is called the iterator cursor, and it is defined as an interface; iterator interface. Whatever the limitations that we have learned, so far the enumeration interface is concerned, it has been addressed into this iterator interface. This interfaces in one sense, it is called the universal because this iterator interface can be applied to many collections like set, list, queue, dequeue.



And also, it can be implemented in the map interface also that we have already studied and we have exercised while we are discussing all these collections in our previous learning. Now, the speculative of these interface is that using this cursor iterator, we can perform read and remove operation. However, we will not be able to perform any place or addition operation. Remove operation is possible but replace or addition operation is not possible. And let us see how this iterator or cursor you can use in your program.

Now, in order to use this cursor, there are three methods that you can rely on. These methods are has next, next, and remove. Remove is basically for removing purpose. And has next or next is basically to check that it has further element or not. And if it has, then it return the next element, actually.

(Refer Slide Time: 09:58)



The slide displays a Java code snippet for an iterator demo. The code is as follows:

```
import java.util.*;

public class TIteratorDemo {
    public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        for (int i = 0; i < 10; i++)
            al.add(i);

        System.out.println(al);
        Iterator itr = al.iterator();
        // The cursor at the beginning of the first element.
        while (itr.hasNext()) { // iterate over each element in al
            int i = itr.next(); // Read the current element.
            System.out.print(i + " ");
            if (i % 2 != 0)
                itr.remove(); // Removing odd elements
        }
        System.out.println();
        System.out.println(al);
    }
}
```

The slide also features a small video inset of a man in the bottom right corner and logos for NPTEL and IIT Kharagpur at the bottom.

So let us have some illustration of this interface. And one illustration I gave it to you so that you can understand about it. This program gave a quick demo about this cursor iterator and we create one collection. The name of the collection is array list, is a array list of integer. And we create the array list here using this method. So this basically, this are list contents 10 different elements in it.

This is a usual print ln statement, overall printing of this statement. But we are interested to traverse it using our cursor iterator. So this is important code that you can think about. We create

an iterator. It is always we have to create an iterator object for that. So this is called for the array list collection, this is the method that we have used, the factory method that you should use to call this iterator. So this iterator is easy to traverse.

Initially, it is starting from this one, now here, has next so it takes that it has the element. So it is basically currently pointing at 0 and then it return the 0th element. So this is basically integer value it is there because we have defined for integer collection, so it written integer. And then, i store the temporary value it is be. What it returns, it store it there and it print it. That is fine. So this basically traverse each element and print it there.

Now, what we are doing here, if i dip 2 equals to 0, that means if the number is even, then we just remove it. So while we are traversing, we are removing it, and then after the entire traversal is over, the array list collection will obtain after removing all the even numbers. So if we print here, you will see it basically gives the modified array list after removing all the even numbers in this.

So this is one example that you can check that okay, it basically, how these cursor work for the collection array list. So this this kind of illustration can be extended to other collection also, which we have already exercised there. I am mentioning how they are, basically merits and demerits of it.

(Refer Slide Time: 12:04)

**Example 60.3 : Add or remove while using iterator**

```
import java.util.*;
public class ErrorWithIterator {
    public static void main(String args[]){
        ArrayList<String> books = new ArrayList<String>();
        books.add("");
        books.add("++");
        books.add("Java");
        for(String obj : books){
            System.out.println(obj);
            // we are adding element while iterating list
            books.add("C");
        }
        Iterator itr = books.iterator();
        while (itr.hasNext()) {
            String b = itr.next();
            System.out.println(b + " ");
            books.add("python"); // you cannot do it!
            books.remove("C"); // You cannot do it!!
        }
    }
}
```

**Be careful.**  
You cannot add or remove elements to the collection while using iterator over it.  
Hence, it will give a run-time exception.

NPTEL Online Certification Courses  
IIT Kharagpur

Now, here is another example. As I have mentioned, that this kind of iterator class cannot be used in order to replace or add some new elements. So this is the one example that you can see for this purpose. This program give a demo. While we, if we want to attempt at some elements or replace some elements, while we are traversing using iterator cursor.

So for this purpose, let us create a collection of a user-defined class say, books. Assume that books is already defined somewhere. So this program, assume that books class is declared there, or it can be a string also. We can create a collection of string also, fine. Otherwise, if you use a user-defined, you can use that also no issue.

So this basically, we create. We insert three different elements into this collection books, which is of type string. Now, using for each loop and you can see for which loop is the one also default cursor. Basically, it also traverse just like while loop for loop and this is for each loop. So it basically, for each object belongs to this collection books, it basically moves or basically visits. It visits means prints, and then while we are doing, we can add one object, say, this one; so we can do it also.

So this is possible because we can do a usual addition while we are using for. So it basically, each time, we can add one elements here, new elements will be added and the collection allows duplicate entry. So it is possible. Now, let us come to the discussion of the same thing but using our cursor iterator.

(Refer Slide Time: 14:12)

The slide displays a Java code snippet and a warning message. The code is as follows:

```
import java.util.*;
public class ErrorWithIterator {
    public static void main(String args[]){
        ArrayList<String> books = new ArrayList<String>();
        books.add("");
        books.add("C++");
        books.add("Java");

        for(String obj : books) {
            System.out.println(obj);
            //We are adding element while iterating list
            books.add("C");
        }

        Iterator itr = books.iterator();
        while (itr.hasNext()) {
            String b = itr.next();
            System.out.println(b + " ");
            books.add("Python"); // you cannot do it!
            books.remove("C"); // You cannot do it!!
        }
    }
}
```

Be careful.

You cannot add or remove elements to the collection while using iterator over it.

Hence, it will give a run-time exception.

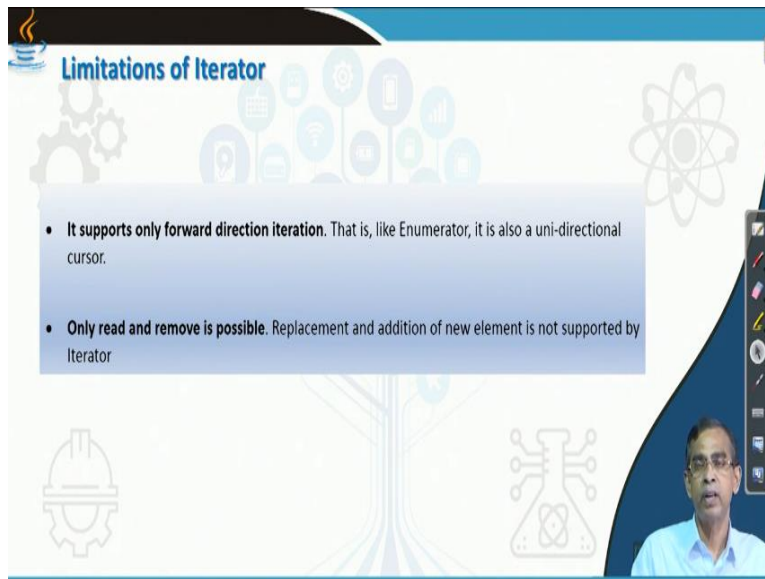
The slide also features the NPTEL logo and the text 'NPTEL Online Certification Courses IIT Kharagpur' at the bottom.

Now, this point is interesting to note. What it basically does for, does here. So we create an iterator for this collection, books. So it check that okay, it is in starting from this, so we can do it. Now, it basically start, use the new collection here because in this case, the collection will be this plus all the elements are added. So that also you should note it. So when you see the output, you can get it.

But anyway, this basically program is to illustrate some limitation of these cursor interface rather not limitation, it basically is a constant we can say. So it basically traverse it. So this is not an issue, this is also fine. It basically traverse it print also. But whenever it comes here, the add method or this replace method instead of remove, we can say replace. It is basically replace method. We can use a replace by; remove c means, it basically replace. So it basically, c is removed.

Now here, if you see, this basically point out an error. This because it cannot, this iterator can or should, it cannot allow you to do addition into while it is traversing. Now, this remove is fine, it can do it. But if we replace this method by replace, I mean, if you write another call books dot replace c, then c by another string, then you can see it is also not permissible. So here, you have to be careful about that this kind of iterator does not allow to add or remove any elements while it basically this iterator works. Or it, you traverse using this iterator.

(Refer Slide Time: 16:04)



**Limitations of Iterator**

- It supports only forward direction iteration. That is, like Enumerator, it is also a uni-directional cursor.
- Only read and remove is possible. Replacement and addition of new element is not supported by Iterator

So this is the only limitation that it can do it. This is just like enumerator it always allow you for forward direction only, not that backward direction or to and fro direction. Only read and remove is possible, but no new element can be added in this method.

(Refer Slide Time: 16:22)



**ListIterator Interface**

NPTEL Online Certification Courses  
IIT Kharagpur

Now, here, let us consider this list iterator interface. List iterator interface is the another interface which is more smarter interface of what is called the cursor than the other two cursors, like enumerator and list interface.

(Refer Slide Time: 16:47)

### ListIterator interface

- It is only applicable for List collection implemented classes like ArrayList, LinkedList, etc.
- It provides bi-directional iteration.
- This cursor has more functionality than iterator.
- ListIterator interface extends Iterator interface. So all three methods of Iterator interface are available for ListIterator.
- In addition, there are six more methods, which are listed next.

Method	Description
<code>public boolean hasNext();</code>	Returns true if the iterator has more elements.
<code>public Object next();</code>	Returns the next element in the iterator.
<code>public void remove();</code>	Remove the next element in the iterator. This method can be called only once per call to next().

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us have the methods which are defined here. Now, regarding this interface, this interface is applicable for any type of list; only applicable for list collection. You can understand list means like array list, link list, and others. So it is basically applicable to link list structure, actually.

And here, you see, all these link list structures which are mentioned in java collection framework, they use double link list. This means that both forward and backward traversal is possible. This means that this cursor allows both bidirectional movement. And it has the same functionality than iterator, means it can remove also while it is traversing. In addition to, it has more functionality that we will discuss.

Now, list iterator interface extend the iterator interface. So all the methods those are declared there is also available to it. In addition to, there are six more method, which also possible. These are the method as we have already experienced with iterator, cursor also it is applicable.

(Refer Slide Time: 17:57)

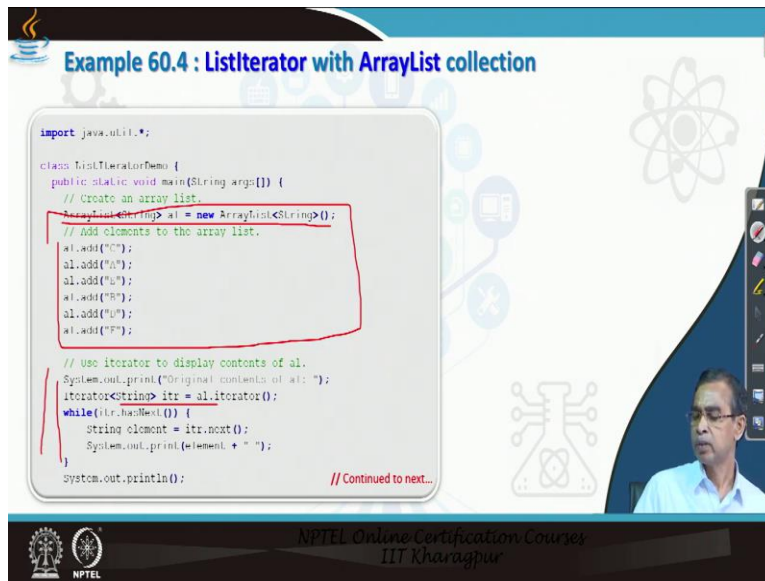
Method	Description
<code>void add(E obj)</code>	Inserts <i>obj</i> into the list in front of the element that will be returned by the next call to <code>next()</code> .
<code>default void forEachRemaining(Consumer&lt;? super E&gt; action)</code>	The action specified by <i>action</i> is executed on each unprocessed element in the collection. (Added by JDK 8.)
<code>boolean hasNext()</code>	Returns <b>true</b> if there is a next element. Otherwise, returns <b>false</b> .
<code>boolean hasPrevious()</code>	Returns <b>true</b> if there is a previous element. Otherwise, returns <b>false</b> .
<code>E next()</code>	Returns the next element. A <code>NoSuchElementException</code> is thrown if there is not a next element.
<code>int nextIndex()</code>	Returns the index of the next element. If there is not a next element, returns the size of the list.
<code>E previous()</code>	Returns the previous element. A <code>NoSuchElementException</code> is thrown if there is not a previous element.
<code>int previousIndex()</code>	Returns the index of the previous element. If there is not a previous element, returns <code>-1</code> .
<code>void remove()</code>	Removes the current element from the list. An <code>IllegalStateException</code> is thrown if <code>remove()</code> is called before <code>next()</code> or <code>previous()</code> is invoked.
<code>void set(E obj)</code>	Assigns <i>obj</i> to the current element. This is the element last returned by a call to either <code>next()</code> or <code>previous()</code> .

Now let us see what are the additional methods, which are defined in this class specially. So these are the methods which are declared here. It add, it basically allows add, whereas the simple cursor iterator cannot allow. And for each remaining also, it is allowed here; has next and we have has previous. It is basically for bidirectional from to and fro also it is possible. Next method we have already declared, next index also, it basically return if you want to see.

The previous element, it return what is the previous element; index of the previous element. Remove, and set, here set means it is basically replace. Now, so this iterator compared to the simple cursor iterator, provides more functionality and therefore, more flexibility to the programmer. So it supports more flexibility that is why many people, only it has limitation that it cannot be applied to all cursor, except the array list, link list cursor, you can apply it.



(Refer Slide Time: 18:51)



**Example 60.4 : ListIterator with ArrayList collection**

```
import java.util.*;

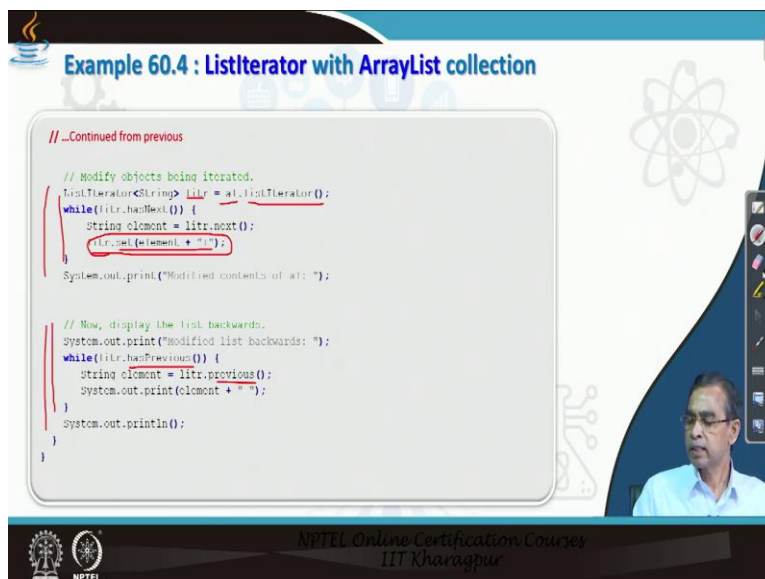
class ListIteratorDemo {
    public static void main(String args[]) {
        // Create an array list.
        ArrayList<String> al = new ArrayList<String>();
        // Add elements to the array list.
        al.add("C");
        al.add("A");
        al.add("B");
        al.add("D");
        al.add("E");
        al.add("F");

        // Use iterator to display contents of al.
        System.out.print("Original contents of al: ");
        Iterator<String> itr = al.iterator();
        while (itr.hasNext()) {
            String element = itr.next();
            System.out.print(element + " ");
        }
        System.out.println();
    }
} // Continued to next...
```

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us consider this as an example that we can apply. So we create a collection, let it be array list, and add some element. So initially, the collection is loaded. Now, we are using one iterator. This is the iterator, has next, and this is basically simple, using our cursor iterator. Now, we can apply the same. We can apply the same okay for the same purpose. We can apply the list iterator method also.

(Refer Slide Time: 19:25)



**Example 60.4 : ListIterator with ArrayList collection**

```
// ...Continued from previous

// Modify objects being iterated.
ListIterator<String> litr = al.listIterator();
while (litr.hasNext()) {
    String element = litr.next();
    litr.set(element + "1");
}
System.out.print("Modified contents of al: ");

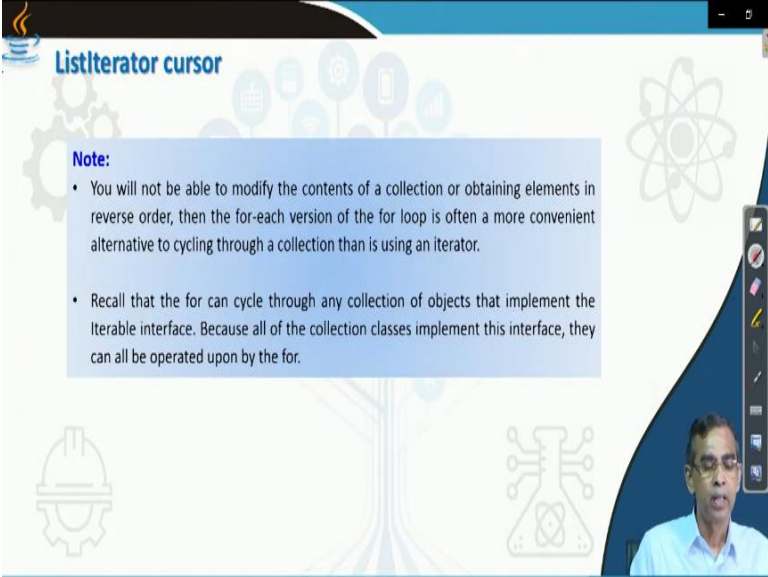
// Now, display the list backwards.
System.out.print("Modified list backwards: ");
while (litr.hasPrevious()) {
    String element = litr.previous();
    System.out.print(element + " ");
}
System.out.println();
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

Here, we are using for the same collection, the list iterator. So for which this method needs to be called on this cursor collection. This is the iterator and here we are traversing. It is basically same, but here, you see, at the time of iterator, we basically can replace element. So we are replacing here this one. So this is allowed. It will not give any error.

And also, you see, we can have some backward. For this backward, we can call hash previous, and then it is a previous element. So from back to front direction also, we can move it. So both way direction, this interface can allow you. So this is, so this basically shows that how it is more advantageous than the other cursor that we have discussed about it.

(Refer Slide Time: 20:15)



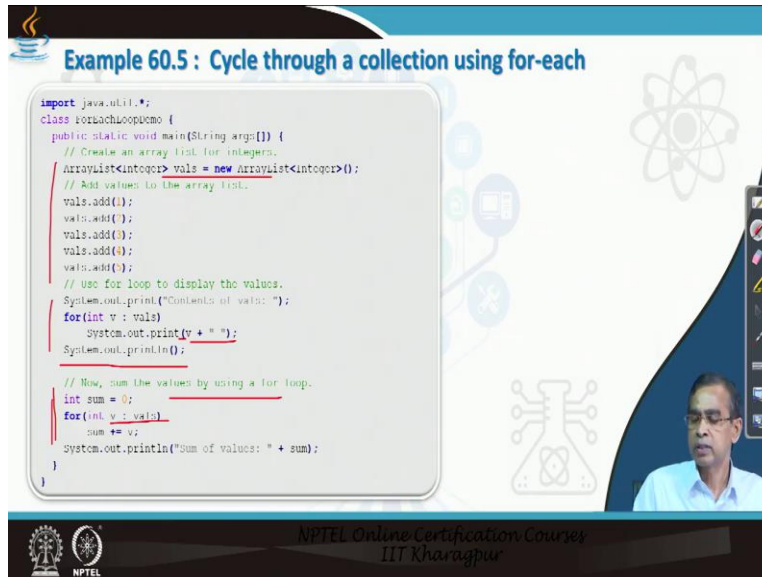
The screenshot shows a presentation slide with the title "ListIterator cursor" in the top left. A blue box labeled "Note:" contains two bullet points:

- You will not be able to modify the contents of a collection or obtaining elements in reverse order, then the for-each version of the for loop is often a more convenient alternative to cycling through a collection than is using an iterator.
- Recall that the for can cycle through any collection of objects that implement the Iterable interface. Because all of the collection classes implement this interface, they can all be operated upon by the for.

A small video inset of a man in a light blue shirt is visible in the bottom right corner of the slide.

So this is more useful cursor than the other two cursor, other two cursors, namely enumeration and then iterator that we have covered.

(Refer Slide Time: 20:35)



**Example 60.5 : Cycle through a collection using for-each**

```
import java.util.*;
class ForEachLoopDemo {
    public static void main(String args[]) {
        // Create an array list for integers.
        ArrayList<Integer> vals = new ArrayList<Integer>(0);
        // Add values to the array list.
        vals.add(1);
        vals.add(2);
        vals.add(3);
        vals.add(4);
        vals.add(5);
        // Use for loop to display the values.
        System.out.println("Contents of vals: ");
        for(int v : vals)
            System.out.println(v + " ");
        System.out.println();

        // Now, sum the values by using a for loop.
        int sum = 0;
        for(int v : vals)
            sum += v;
        System.out.println("Sum of values: " + sum);
    }
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

Now, there is one more example. It is basically called the cycling through a collection using for each loop. So this example tells about that how the cycling, cycling means if you want to end, it can go to the previous and actually cyclic in nature. So this is the one array list collection. And here, using for each, you can just see and come to whenever you go, it will basically go to the end actually.

And then, again if you, this basically calculate the different values of this one, integer v values, and then the result is basically. It basically print and it basically repeat this one. So here, use for loop to display the value; here, use the same loops to add the value of each interface. So you can cyclic it because we have started it and again come back to 0 and then you can do it; so cyclic. And using list iterator also, you can do the cycling; that is also possible.

(Refer Slide Time: 21:33)

### Example 60.5 : Cycle through a collection using for-each

**Note:**

- Clearly the three methods that `ListIterator` inherits from `Iterator` (`hasNext()`, `next()`, and `remove()`) do exactly the same thing in both interfaces. The `hasPrevious()` and the `previous()` operations are exact analogues of `hasNext()` and `next()`. The former operations refer to the element before the (implicit) cursor, whereas the latter refer to the element after the cursor. The `previous()` operation moves the cursor backward, whereas `next()` moves it forward.
- `ListIterator` has no current element; its cursor position always lies between the element that would be returned by a call to `previous()` and the element that would be returned by a call to `next()`.
- Please note that initially any iterator reference will point to the index just before the index of first element in a collection.
- We don't create objects of `Enumeration`, `Iterator`, `ListIterator` because they are interfaces. We use methods like `elements()`, `iterator()`, `listIterator()` to create objects. These methods have anonymous inner classes that extends respective interfaces and return this class object. This can be verified by below code.

Now, what we can summarize from the previous discussion is that the enumerator interface one way; then cursor iterator, that is also one way. Cursor iterator on the contrast to enumerator, it can allow certain CRUD operation, namely, remove, but it cannot allow add and then replace operation, whereas the list iterator can be in two ways and allows all sort of modified operation that is possible.

(Refer Slide Time: 22:08)

### Example 60.6 : Iterators references of different types to the same collection

```
import java.util.Enumeration;
import java.util.Iterator;
import java.util.ListIterator;
import java.util.Vector;

public class JavaCuroorstest {
    public static void main(String[] args) {
        Vector v = new Vector();

        // Create three Iterators
        Enumeration e = v.elements();
        Iterator itr = v.iterator();
        ListIterator ltr = v.listIterator();

        // Code to use the Iterators
    }
}
```

Now, this basically is an example of list iterator with respect to vector. Vector, okay; let us see, the legacy class vector, we can have the enumeration set for this. So here, basically, this is a iterator reference of different types of the same collection. I want to see how for the same collection and the different iterator can be called.


Here, we can see the vector. Vector has some collection list iterator cursor, enumeration cursor. We want to apply all this three cursor on the vector collection. Now, vector is one element vector is the one collection which basically list iterator also supports. In addition to array list and link list, vector also because vector stored in a dynamic position. So it can also.

Now, this example shows on the vector how it can be applied. So it is enumeration e, iterator this one. And then we can apply list iterator over the vector v. And here, we can apply on the enumerator. And here, we can apply on the iterator. Then we can use whatever the way we can iterate. For each iterator, you can traverse it.

So this example shows that this iterator, the three different iterator that we have learned, can be applied to the same. That is a case, of course, but only in the context of vector, it can be applied.

(Refer Slide Time: 23:37)



 **Example 60.7 : Iterators references of different types to the same collection**

It is the most powerful iterator but it is only applicable to List implemented classes, so it is not a universal iterator.


```
public class Employee {
    private int empid;
    private String ename;
    private String designation;
    private double salary;


    public Employee(int empid, String ename, String designation, double salary) {
        this.empid = empid;
        this.ename = ename;
        this.designation = designation;
        this.salary = salary;
    }

    public int getEmpid() {
        return empid;
    }

    public String getEname() {
        return ename;
    }

    public String getDesignation() {
        return designation;
    }
}
// Continued to next...
```



 NPTEL Online Certification Courses  
IIT Kharagpur

Now, there are certain limitations of the list iterators that we want to discuss about it. The limitations is that this iterator is not ((thread))(23:50), say, that is the one. But it is the powerful iterator, but only it is applicable to list, list that we have told. In addition to this, vector also can be. And that is why this can be consider as a more useful iterator.

Now, this iterator also can be applied to user-defined object as a collection. And here, we can define one object of, one class we are defining. The name of the class is employee. And this is the usual definition with constructor, getter, and setter methods. Now, let us first discuss the class declaration, and then we will come to the creation of a collection of these objects, and then iterator over it.

(Refer Slide Time: 24:34)

The slide displays a code editor with the following Java code:

```
// -- Continued from previous

public double getSalary() {
    return salary;
}

//this shows how to print an element of custom type using toString()
@Override
public String toString(){
    return empid + "\t" + name + "\t" + designation + "\t" + salary;
}

import java.util.*;

public class employees implements Iterable{
    private List<Employee> emps = null;

    public Employee(){
        emps = new ArrayList<Employee>();
        emps.add(new Employee(101,"Sam","Professor", 700000));
        emps.add(new Employee(102,"Rahim","Engineer", 300000));
        emps.add(new Employee(103,"Jonny","Doctor", 350000));
    }

    // Continued to next...
```

The slide also features the NPTEL logo and the text 'NPTEL Online Certification Courses IIT Kharagpur' at the bottom.

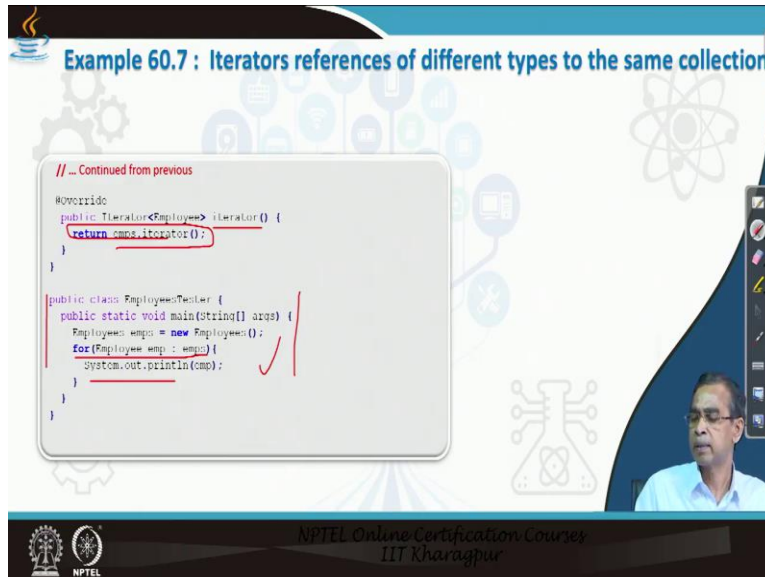
So this basically completes the declarations of class. Then here, we just create you see, we want to make this employee class. So that it is iterable, that is one important thing that we have to then only you can iterate over it. And we create a collection, we can create a collection is basically list type.

So employee initially it is learn, then we can create some objects of it and add into this collection employee. So this employee then contains so many objects with this one. Now, you can again use the map also there you can use it and then do it. But anyway, we are discussing this thing in the context of simple list.

So list is an interface for which we are using basically is a collection as a array list here. So array list is the collection whose type is list because it is the interface because array list implements a list so that how you can do that. So this basically, essentially, employee is a collection of array list and the array list contains few objects which are here. Now let us see how we can iterate over these objects.



(Refer Slide Time: 25:55)



The slide displays a code editor window with the following Java code:

```
// ... Continued from previous
@Override
public Iterator<Employee> iterator() {
    return emp.iterator();
}

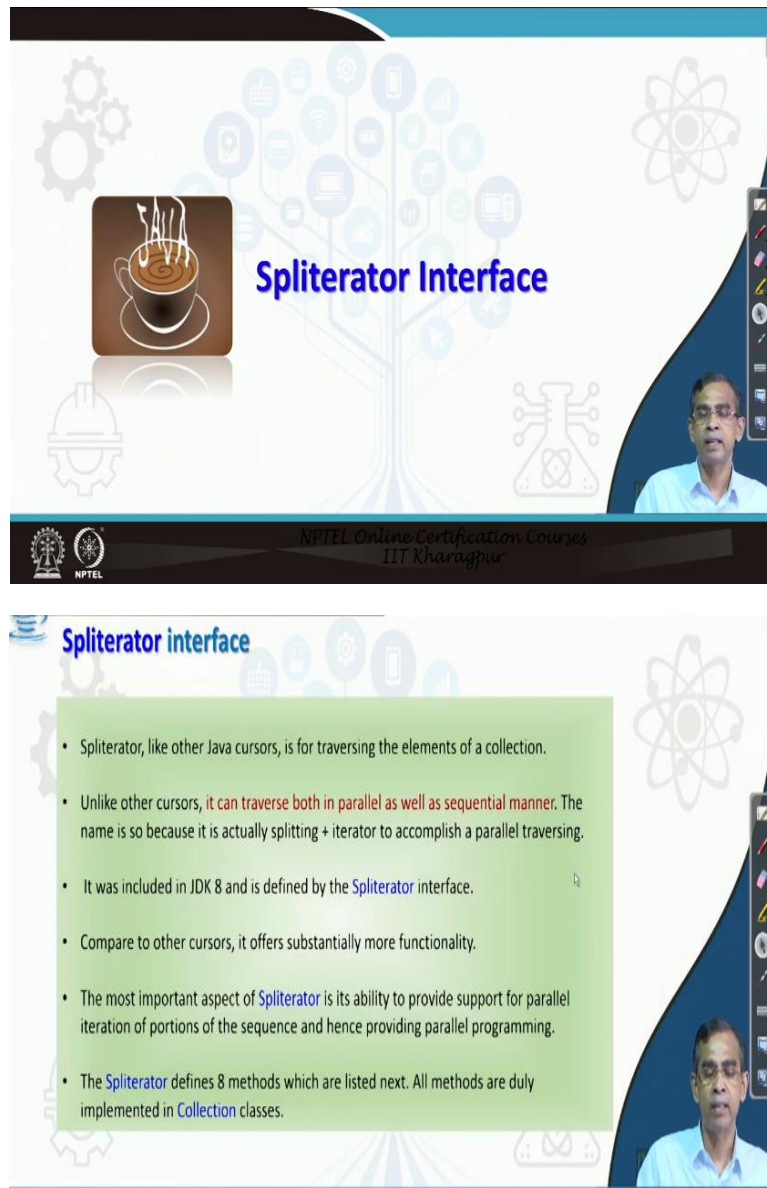
public class EmployeeTester {
    public static void main(String[] args) {
        Employee emp = new Employee();
        for(Employee emp : emp) {
            System.out.println(emp);
        }
    }
}
```

The code is annotated with red lines and a checkmark. The `return emp.iterator();` line is underlined. The `new Employee()` line is underlined. The `for(Employee emp : emp)` line is underlined. A red checkmark is placed to the right of the `for` loop. The slide also features the NPTEL logo and the text 'NPTEL Online Certification Course IIT Kharagpur' at the bottom.

So I want to say that this iterator can be applied to any type of elements in the collection, and if you want to apply your user-defined, then you have to implement the iterable. That is the only thing that you have to do it. Then only you can apply, otherwise, you will not be applied. Now here, you can if you want, then you can override this iterator method according to your own customizations. So basically, we are just overwriting this customization.

Then in this method, we are just using either for each loop or a simple cursor iterator or list iterator you can apply to traverse all the elements here. It is now possible. If you do not implement using this one, you will not be able to do that. That is what I want to mention in this example. So you can test it and you can do certain modifications so that whether it works or not, then you will be able to realize that how these things works.

(Refer Slide Time: 26:44)



The top screenshot shows a slide titled "Spliterator Interface" with a coffee cup icon. The bottom screenshot shows a slide titled "Spliterator interface" with a list of bullet points.

- Spliterator, like other Java cursors, is for traversing the elements of a collection.
- Unlike other cursors, it can traverse both in parallel as well as sequential manner. The name is so because it is actually splitting + iterator to accomplish a parallel traversing.
- It was included in JDK 8 and is defined by the Spliterator interface.
- Compare to other cursors, it offers substantially more functionality.
- The most important aspect of Spliterator is its ability to provide support for parallel iteration of portions of the sequence and hence providing parallel programming.
- The Spliterator defines 8 methods which are listed next. All methods are duly implemented in Collection classes.

Now, let us come to the spliterator which is the another what is called the new addition of the java developer from the Java Development Kit. It is called the spliterator. It is just like other java cursor to traverse the element of a collection. But only difference is that, like other collection, it basically allow parallel traversing and all parallel traversing in a sequential manner.

So it is basically if you want to split a list, and then traverse all these at in a parallel fashion, then that, this cursor you can use it. That is why it name is spliterator iterator. It basically comes from splitting an iterator. So this way it basically allows parallel traversing.

So it was introduced recently in JDK 8 and defined by an interface is called a spliterators interface, then it implements in all other collection classes those are basically defined there. So compared to the other cursor, it has substantially more functionality, it is gives more utility functions, we can say the most important aspect of spliterator is ability to provide support for parallel processing.

That is why it is used and, otherwise, it basically does the same thing as the other iterator does it. And you can again repeat all the programs or illustration, illustrative program that we have given in the context of this iterators then you can get its advantages.

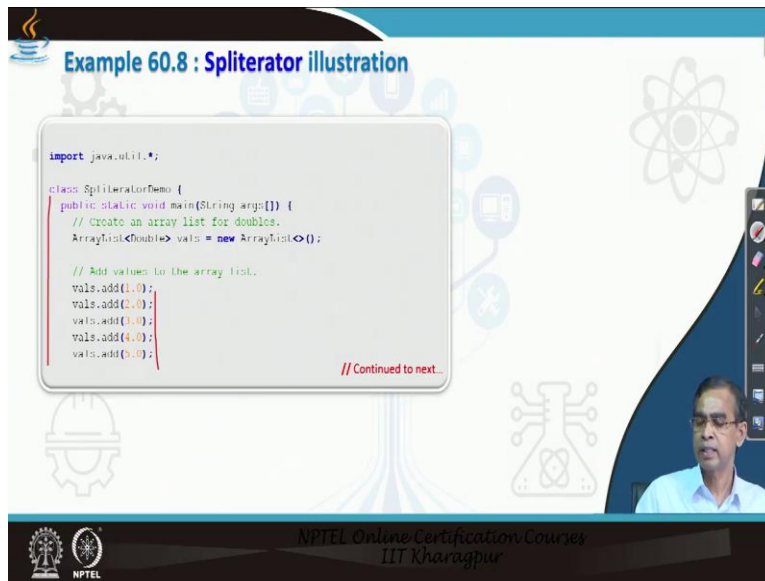
(Refer Slide Time: 28:30)

Method	Description
<code>int characteristics()</code>	Returns the characteristics of the invoking spliterator, encoded into an integer.
<code>long estimateSize()</code>	Estimates the number of elements left to iterate and returns the result. Returns <code>Long.MAX_VALUE</code> if the count cannot be obtained for any reason.
<code>default void forEachRemaining(Consumer&lt;? super T&gt; action)</code>	Applies <i>action</i> to each unprocessed element in the data source.
<code>default Comparator&lt;? super T&gt; getComparator()</code>	Returns the comparator used by the invoking spliterator or <code>null</code> if natural ordering is used. If the sequence is unordered, <code>IllegalStateException</code> is thrown.
<code>default long getExactSizeIfKnown()</code>	If the invoking spliterator is sized, returns the number of elements left to iterate. Returns <code>-1</code> otherwise.
<code>default boolean hasCharacteristics(int m)</code>	Returns <code>true</code> if the invoking spliterator has the characteristics passed in <i>m</i> . Returns <code>false</code> otherwise.
<code>boolean tryAdvance(Consumer&lt;? super T&gt; action)</code>	Executes <i>action</i> on the next element in the iteration. Returns <code>true</code> if there is a next element. Returns <code>false</code> if no elements remain.
<code>Spliterator&lt;T&gt; trySplit()</code>	If possible, splits the invoking spliterator, returning a reference to a new spliterator for the partition. Otherwise, returns <code>null</code> . Thus, if successful, the original spliterator iterates over one portion of the sequence and the returned spliterator iterates over the other portion.

So this is about a different other extra method by which you can apply to this spliterator and you can get it. Here, we have mentioned few method is a characteristics, so it returns the specific properties of the iterator that it can return. And there are few method, estimates size, all these things is basically regarding parallel version, parallel traversal, all those things is required.

Those description you can have it from here and you can practice it in your program and calling these methods. And then you can get it. And then you see that basically, it has more powerful features than the other three cursors that we have discussed.

(Refer Slide Time: 29:10)



**Example 60.8 : Spliterator illustration**

```
import java.util.*;

class SpliteratorDemo {
    public static void main(String args[]) {
        // Create an array list for doubles.
        ArrayList<Double> vals = new ArrayList<>();

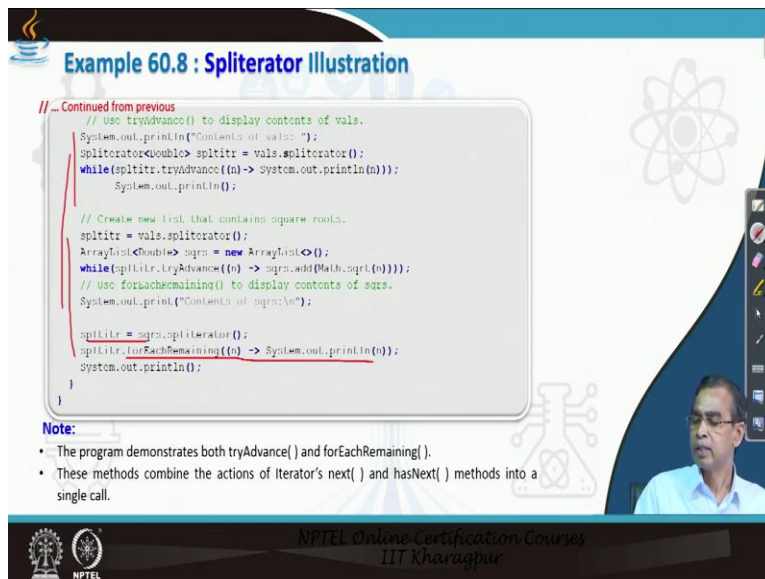
        // Add values to the array list.
        vals.add(1.0);
        vals.add(2.0);
        vals.add(3.0);
        vals.add(4.0);
        vals.add(5.0);

        // Continued to next...
    }
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

So applying these things is very similar. As an example I have given here, it basically does not give that much flavor because it is only a few elements are there. If it is large element, and then split it, and then apply for each portion, then you will be able to see that how it works better. So that you can try with another program.

(Refer Slide Time: 29:31)



**Example 60.8 : Spliterator Illustration**

```
// ... Continued from previous
// Use tryAdvance() to display contents of vals.
System.out.println("Contents of vals: ");
Spliterator<Double> splitr = vals.spliterator();
while(splitr.tryAdvance((n) -> System.out.println(n)))
    System.out.println();

// Create new list that contains square roots.
splitr = vals.spliterator();
ArrayList<Double> sqrs = new ArrayList<>();
while(splitr.tryAdvance((n) -> sqrs.add(Math.sqrt(n))))
    // Use forEachRemaining() to display contents of sqrs.
    System.out.print("Contents of sqrs:\n");

splitr = sqrs.spliterator();
splitr.forEachRemaining((n) -> System.out.println(n));
System.out.println();
}
```

**Note:**

- The program demonstrates both tryAdvance() and forEachRemaining().
- These methods combine the actions of Iterator's next() and hasNext() into a single call.

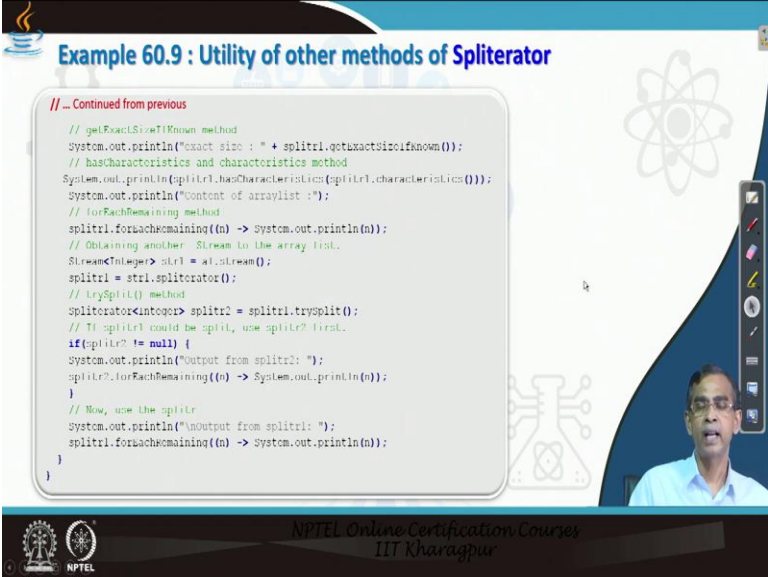
NPTEL Online Certification Courses  
IIT Kharagpur

Here, we have done the same thing actually. Here, we basically split it, and then on splitting part, we can call this and then how it work it is there. And here, we again using, in addition to this

spliterator, we are using for each with the lambda expression also to print it. So basically, in that sense, it can be combined with the own iteration. In addition to other iteration also, it can be mixed. That means, it is basically a hybrid cacterization it can be possible with this one.

So this example basically shows how it is possible that spliterator can be combined with for each or for each remaining portion it is. And there are some other methods which we have listed there and how they can be exercised. This program basically illustrate this concept. You can run this program and you can check that how the parallel traversing is possible with this cursor, the spliterator.

(Refer Slide Time: 30:39)



The slide displays a code editor window with the following Java code:

```
// ... Continued from previous
// getExactSizeIfKnown method
System.out.println("exact size : " + splitr1.getExactSizeIfKnown());
// hasCharacteristics and characteristics method
System.out.println(splitr1.hasCharacteristics(splitr1.characteristics()));
System.out.println("content of arraylist :");
// forEachRemaining method
splitr1.forEachRemaining((n) -> System.out.println(n));
// Obtaining another Stream to the array list.
Stream<Integer> str1 = a1.stream();
splitr1 = str1.splitIterator();
// trySplit() method
Spliterator<Integer> splitr2 = splitr1.trySplit();
// If splitr1 could be split, use splitr2 first.
if(splitr2 != null) {
System.out.println("Output from splitr2: ");
splitr2.forEachRemaining((n) -> System.out.println(n));
}
// Now, use the splitr
System.out.println("\nOutput from splitr1: ");
splitr1.forEachRemaining((n) -> System.out.println(n));
}
```

The slide also features the NPTEL logo, the text "NPTEL Online Certification Courses IIT Kharagpur", and a small video inset of a man in the bottom right corner.

So this program, you can run it and you can check that how the different features of the spliterator can be enjoyed in a program.

(Refer Slide Time: 30:48)

**Utility of other methods of Spliterator**

Have a look at `tryAdvance()` method. It performs an action on the next element and then advances the iterator. It is shown here:

```
boolean tryAdvance(Consumer<? super T> action)
```

Here, action specifies the action that is executed on the next element in the iteration and Consumer is a functional interface that applies an action to an object. It is a generic functional interface declared in `java.util.function`. It has only one abstract method, `accept()`, which is shown here:

```
void accept(T objRef) // Here T is type of object reference.
```

For implementing our action, we must implement `accept` method. To implement `accept` method, here we use lambda expression. This will be more clear from below example.

How to use `Spliterator` with `Collections`: Using `Spliterator` for basic iteration tasks is quite easy, simply call `tryAdvance()` until it returns false.

NPTEL Online Certification Course  
IIT Kharagpur

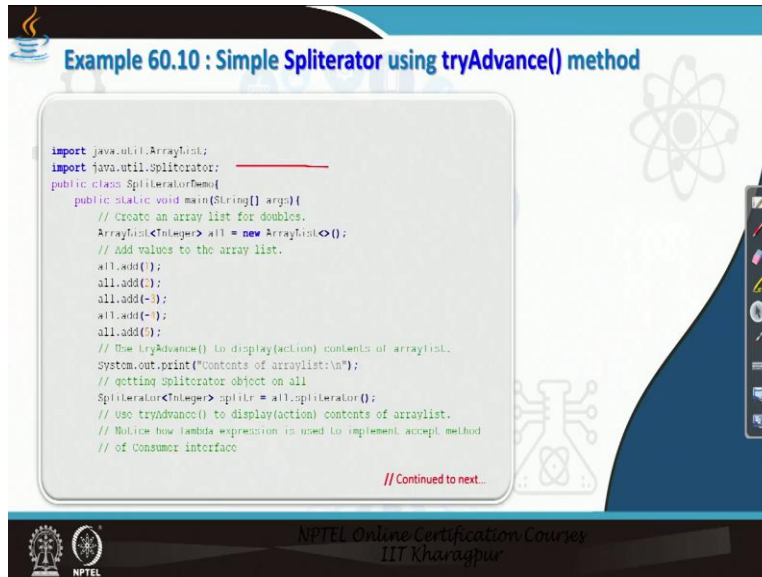
So this basically has that two methods, especially very important in this iterator is called the try advance and accept. Here, the try advance perform an action on the next element and then advance the iterator. Whatever that performance means is, addition, while all modification, whatever you can do that, you can do that. And it can also allow you to perform another extra operation. It is generic and functional interface declared in java dot util function.

One method is called abstract method. It is for implementing any action of your own that you want to do on a particular collection. So that action, `accept` method you can overwrite and then accordingly, you can customize whatever the method during your collection that you want to have.

So there are two important features that is basically advanced features by which you can enjoy much more while you are traversing a collection.



(Refer Slide Time: 31:58)



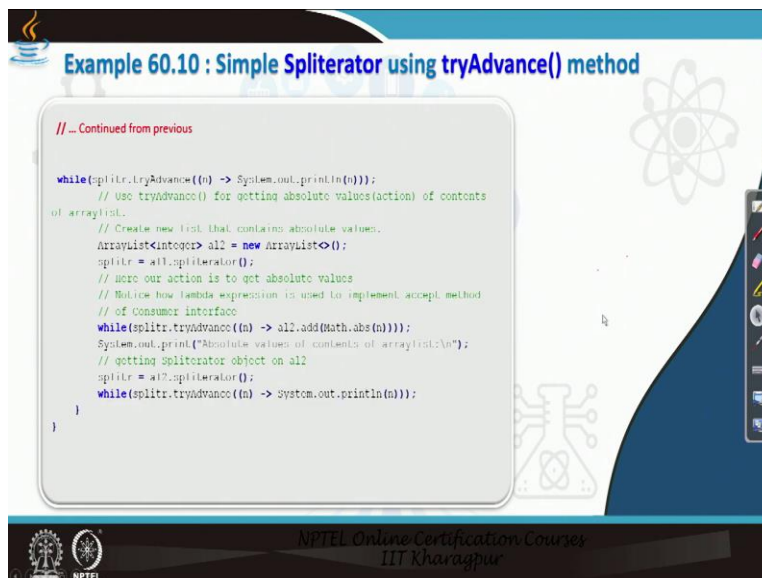
**Example 60.10 : Simple Spliterator using tryAdvance() method**

```
import java.util.ArrayList;
import java.util.Spliterator;
public class SpliteratorDemo {
    public static void main(String[] args) {
        // Create an array list for doubles.
        ArrayList<Integer> all = new ArrayList<>();
        // Add values to the array list.
        all.add(1);
        all.add(2);
        all.add(-1);
        all.add(-1);
        all.add(5);
        // Use tryAdvance() to display(action) contents of arraylist.
        System.out.print("Contents of arraylist:\n");
        // getting Spliterator object on all
        Spliterator<Integer> splitr = all.spliterator();
        // Use tryAdvance() to display(action) contents of arraylist.
        // Notice how lambda expression is used to implement accept method
        // of Consumer interface
        // Continued to next...
```

NPTEL Online Certification Courses  
IIT Kharagpur

So these are the different cursor, different cursor that we have discussed.

(Refer Slide Time: 32:03)



**Example 60.10 : Simple Spliterator using tryAdvance() method**

```
// ... Continued from previous

while(splitr.tryAdvance((n) -> System.out.println(n)));
// Use tryAdvance() for getting absolute values(action) of contents
of arraylist.
// Create new list that contains absolute values.
ArrayList<Integer> all2 = new ArrayList<>();
splitr = all.spliterator();
// here our action is to get absolute values
// Notice how lambda expression is used to implement accept method
// of Consumer interface
while(splitr.tryAdvance((n) -> all2.add(Math.abs(n))));
System.out.print("Absolute values of contents of arraylist:\n");
// getting Spliterator object on all2
splitr = all2.spliterator();
while(splitr.tryAdvance((n) -> System.out.println(n)));
}
```

NPTEL Online Certification Courses  
IIT Kharagpur



### Example 60.10 : Simple Spliterator using tryAdvance() method

```

// ... Continued from previous

while(splitr.tryAdvance((n) -> System.out.println(n)));
// Use tryAdvance() for getting absolute values(action) of contents
of arraylist.
// Create new list that contains absolute values.
ArrayList<Integer> a12 = new ArrayList<>();
splitr = a12.spliterator();
// here our action is to get absolute values
// Notice how lambda expression is used to implement accept method
// of Consumer interface
while(splitr.tryAdvance((n) -> a12.add(Math.abs(n))));
System.out.println("Absolute values of contents of arraylist:\n");
// getting Spliterator object on a12
splitr = a12.spliterator();
while(splitr.tryAdvance((n) -> System.out.println(n)));
}

```

NPTEL Online Certification Courses  
IIT Kharagpur

Here is another example showing the illustration of try advance method that you can check that how it basically do some activity while it is traversing. Basically addition, removal, and all these operations that we have mentioned here. We can generate random number, we can add sum into it; all those things, we can do it.

(Refer Slide Time: 32:24)

### Simple Spliterator using tryAdvance() method

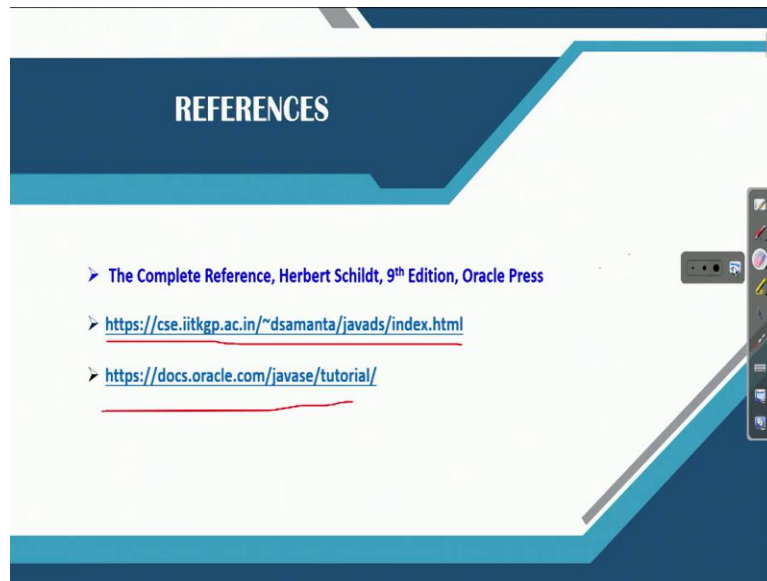
**NOTE:**

- How `tryAdvance()` consolidates the purposes of `hasNext()` and `next()` provided by Iterator into a single method in above example. This improves the efficiency of the iteration process.
- To perform some action on each element collectively, rather than one at a time `Spliterator` provides the `forEachRemaining()` method, it is generally used in cases involving streams. This method applies action to each unprocessed element and then returns.

NPTEL Online Certification Courses  
IIT Kharagpur

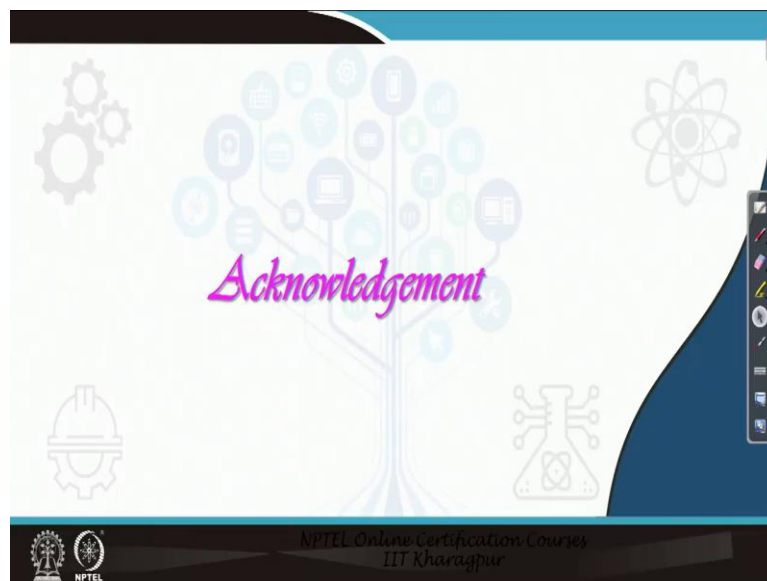
And then accept method is okay, you can overwrite it and then customize this method so that you can get it. And as I told you, spliterator is good that okay, it can be combined with other iterator in addition to this also.

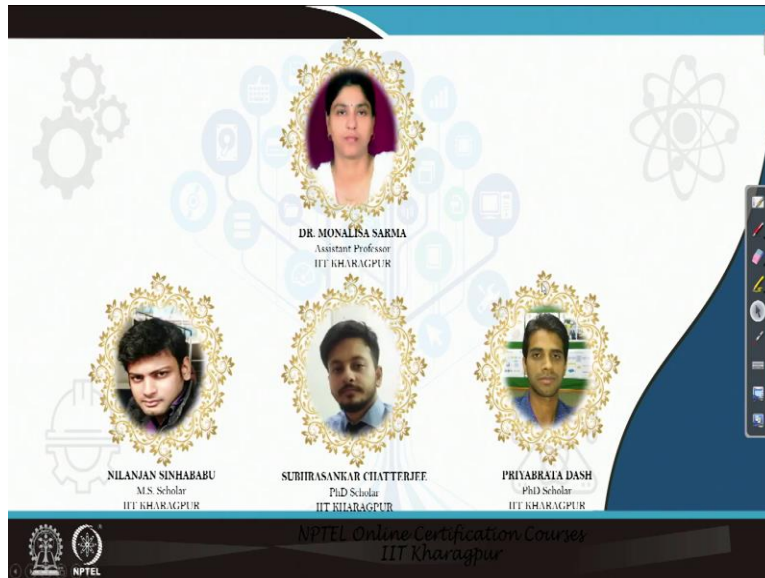
(Refer Slide Time: 32:37)



And for many more detailed discussion, I have already stopped with some more supplementary materials here. So I should advise you to go through. And this is the one document that you can think about so that you can learn much more about this one.

(Refer Slide Time: 33:02)





And so, this is the end of the last video lectures. I want to acknowledge behind the contribution, the different people that I got during the whole process. So it was a very long journey. I started the preparation of this course two years back.

As you can see, there are many materials involved, including the slides, and then contents, the labeling, and so many things are there. So it was a huge job from us. There are many people who supported it. And here are the pillars, we can see behind this course. And Dr. Monalisa here, who helped me to prepare the content in an exhaustive manner so you can see how the coverage of the different topics.

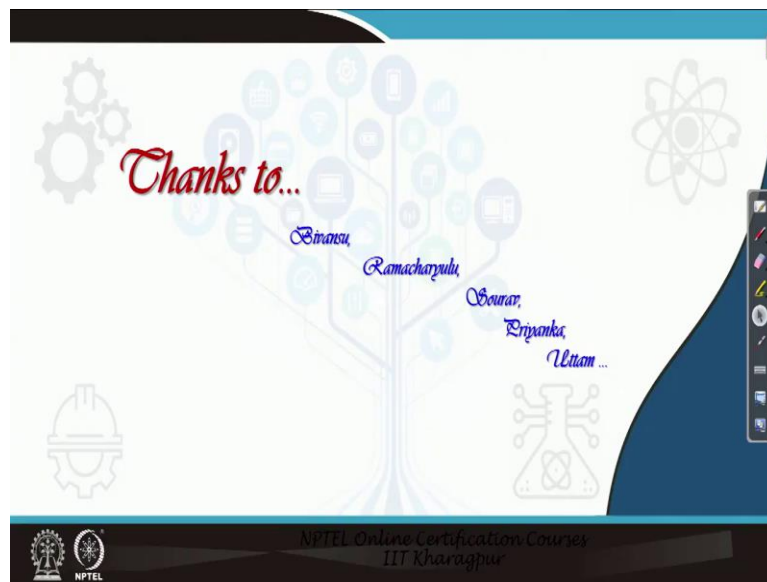
There are many books available related to these java data structure, but no books covered the data structure in that sense, which basically required, which we have taken enough efforts so that essence of data structure using java program can be obtained. This is really a novel effort from our side. So Dr. Monalisa has helped me to prepare many materials in this regard. And then content is enriched with her contribution.

And another people who helped me a lot, preparing particularly the slides is Nilanjan. So you can see so much slides. Although many mistakes, errors are there, definitely it is our limitations, we could not correct. So many errors are there, but we have tried our best to do as much as error-free as possible.

Now, you have also experienced with many programs related to the different discussion. So here Subhrasankar helped me a lot. So whenever I need some programs to be tested or some relevant or some program is required relevant to a particular discussion, so he always came forward and then supported with programs.

And regarding the content management and particularly, weblink and other preparation, HTML document, you can find the link that I have given. So here, Priyabrata helped me a lot. So these are the help that I cannot really, no appreciation is enough to tell thanks. Anyway, so they are the main pillar. And other than, many people also directly, indirectly they are most of my students helped me. And then, I really obey my most sincere gratitude to them.

(Refer Slide Time: 35:41)



And there are many thanks to other people, particularly NPTEL staff, all the lecture video recording you see, we have done it during this lockdown period. It is really a very trying time, very difficult situation because of that COVID pandemic. So there are many NPTEL, those are the people basically help me a lot. And then they took their life in risk and then come to this recording studio and then helped me.

Really they are very sincere. I really thanks to be Bivanshu, Ramu, Saurav, Priyanka, Uttam for their genuine service, and they are disciplined work and systematic way so that this recording is possible today. Thank you very much.

First of all, next, I want to give that thank you to all of you who are attending this course and forwarded your suggestion, feedback, and my effort, my whatever the level that I have put that I think is success only if I see that it really helps you to learn the concept better.

With these things, I want to conclude it. Thank you very much and wish you all the best and fun of reading and learning this concept. Thanks a lot.