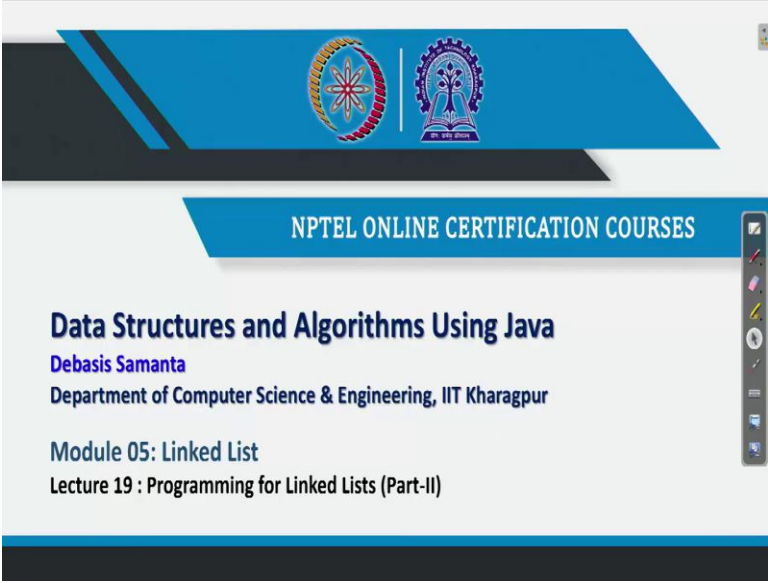


Data Structures and Algorithms using Java
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 19 - Programming for Linked Lists (Part-II)

(Refer Slide Time: 00:29)



The image shows a slide from an NPTEL online certification course. At the top, there is a blue banner with two logos: the Indian Institute of Technology Kharagpur logo on the left and the NPTEL logo on the right. Below the banner, a blue bar contains the text "NPTEL ONLINE CERTIFICATION COURSES". The main content of the slide is as follows:

Data Structures and Algorithms Using Java
Debasis Samanta
Department of Computer Science & Engineering, IIT Kharagpur

Module 05: Linked List
Lecture 19 : Programming for Linked Lists (Part-II)

On the right side of the slide, there is a vertical toolbar with various icons for navigation and editing.

We are discussing programming for linked list. There are few operations and how to define a linked list structures, we have already studied. Hope you are practicing all the programs that we have discussed. So today, we will cover few more operations related to the linked list structures.

Programming overview: Methods

- **Methods**
 - **Insertion operations**
 - Insertion at front
 - Insertion at end
 - Insertion at any position
 - **Traversal**
 - Printing the collection
 - Reversing the ordering of elements
 - **Merging operation**
 - Merging two list into a single list
 - **Deletion operation**
 - Deletion from front
 - Deletion from end
 - Deletion from any position

```
public void insertFront(T data) { ... }
public void insertEnd(T data) { ... }
public void insertKey(T data, T key) { ... }

public void printList() { ... }
public void reverse() { ... }

public void merge(JLinkedList<T> li) { ... }

public T deleteFront() { ... }
public T deleteEnd() { ... }
public void deleteKey(T key) { ... }
```

NPTEL Online Certification Courses
IIT Kharagpur

Now, let us come to the linked list structure again. So, we will discuss about how the other few methods relative (delete), deletion can be covered here. So, today's methods related to deletions, there are three different methods regarding the three different cases. Like deletion at front, we declare this method as delete front. This method does not require any argument but it will return the value of the node that it has deleted successfully.

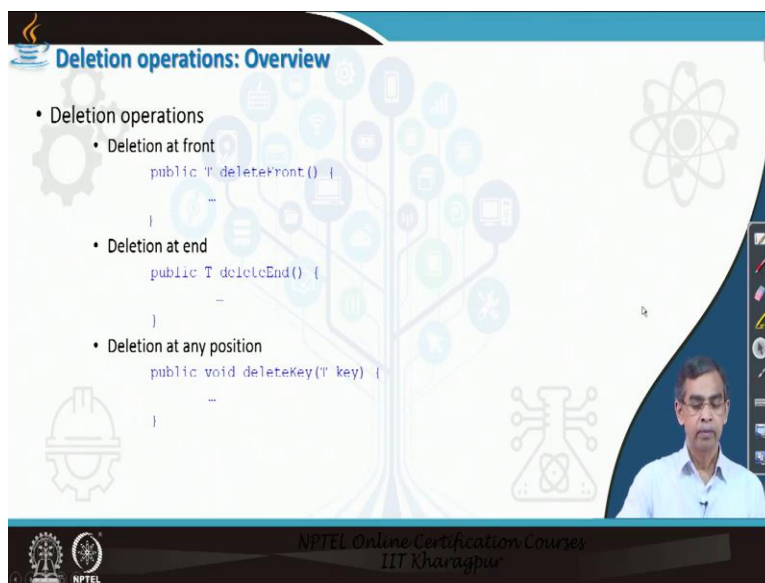
Then, deletion at end. It also does not require any arguments to be passed to this method. It will return the value of the node that it deletes successfully. And finally, we will discuss about delete at any position given a key as an argument that means which elements you want to delete. As it does not, it, it does not require to pass any value, return any value, so its return type is void. So these are the three methods related to deletion operation, will be covered.

(Refer Slide Time: 02:57)



Deletion Operations

NPTEL Online Certification Courses
IIT Kharagpur



Deletion operations: Overview

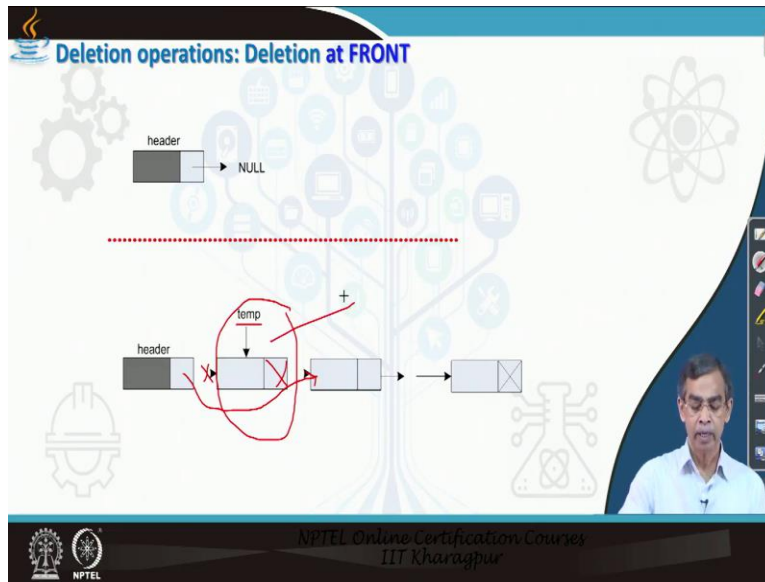
- Deletion operations
 - Deletion at front

```
public T deleteFront() {  
    ...  
}
```
 - Deletion at end

```
public T deleteEnd() {  
    ...  
}
```
 - Deletion at any position

```
public void deleteKey(T key) {  
    ...  
}
```

NPTEL Online Certification Courses
IIT Kharagpur



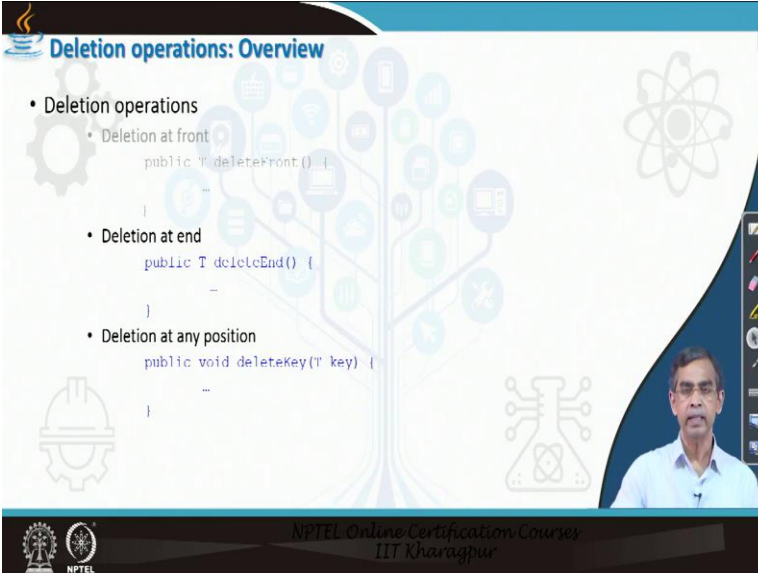
Now, let us discuss all three methods one by one. First we shall start with delete front operation that means deleting a node from the front of a linked list. Now, there are two cases possible in, in this (op), in this operation, mainly deleting something from an empty list. Obviously, there is nothing to delete; so in that case also if you call the method for a linked list, then it will not return anything rather it will return NULL. And the second case, the linked list is non empty. In that case, the deletion will be simply (maniti) manipulating the links. So, first we have to go to the first node. Let the first node be temp.

Then once you know the first node, from the first node we know the link field, link of this node. That mean this basically links the next node. So what we do is that, we will just assign the pointer of this header dot link by the temp dot link. So this basically will set this one. And finally we will mark this link field as null and by this process, this is also, this no more exist. And then we can return this node to the memory bank. So this is the procedure for the deletion at front is concerned. Now let us see, how the method for this link manipulation can be defined.

manipulation is common, it is written in that way actually. Now here, if temp not equals to null, then only we will do. If temp equals to null, return x. x is initially null, as I told you. So for empty list, it will return nothing, null rather. Now here you just see what pointer manipulation we did it.

First we stored the value of the temp node which has to be deleted, so temp dot data equals to x. And then we just change the pointer of the header link by the pointer that the temp node points to its next node. And then we will just print that the element has been deleted successfully and finally we return the value that has been deleted. So this is the method, so for delete front is concerned. I hope it is, you are able to understand it. And, so this is the first method.

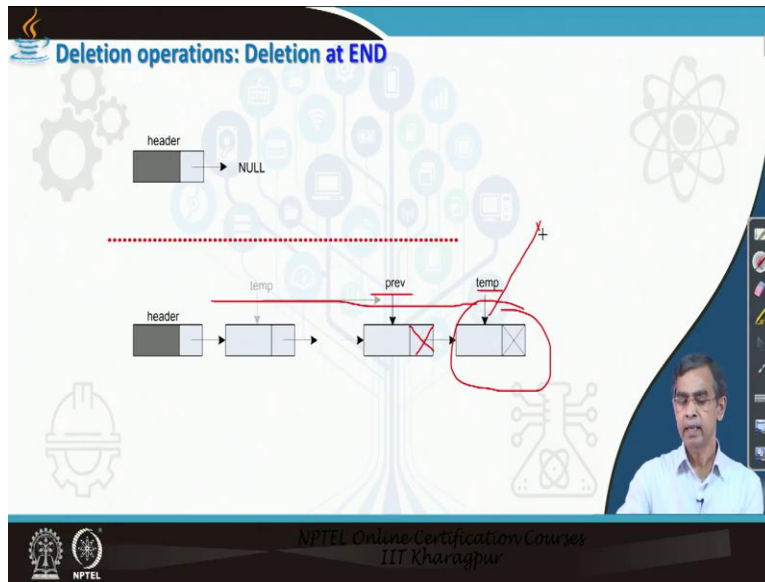
(Refer Slide Time: 06:56)



The slide, titled "Deletion operations: Overview", lists three types of deletion operations with their corresponding code snippets:

- Deletion operations
 - Deletion at front
 - `public T deleteFront() {`
 - `...`
 - `}`
 - Deletion at end
 - `public T deleteEnd() {`
 - `...`
 - `}`
 - Deletion at any position
 - `public void deleteKey(T key) {`
 - `...`
 - `}`

The slide also features the NPTEL logo and the text "NPTEL Online Certification Courses IIT Kharagpur" at the bottom. A small video inset of a presenter is visible in the bottom right corner of the slide.



Now let us consider the another method, delete end. Now delete end method, again there are two situations, empty list or non-empty list. In case of list is empty, we do not have to do anything. In that case if we call this method for a linked list, it will return null. On the other hand, if the list is not empty then what we have to do? Starting from the header, we have to go to the last node. Let the last node be temp, this is the, temp is the last node.

While we are visiting starting from the header to the last node, each moment we should keep a track of its previous node that means the node just one node ahead of the current node. So, this is the previous node. Now this node is required. This is because the deletion will takes place only by (mean), only by assigning null value to this one, and that is enough. So this basically deletes the last node. Now this becomes free, so we should return this node to the memory bank. So, this is the operation delete at end.



(Refer Slide Time: 08:30)

Example 19.2: Declaring a method to delete from END

```
// This part of the program define a method to insert at front

import java.io.*;
import java.util.*;
// Java program to implement a Singly Linked List
public class JLinkedList<T> {
    // Copy the code for the declaration of the class (see Lecture-18, Slide #6)
    // Defining the method to delete a node from the rear
    public T deleteEnd() {
        --
    }
} // End of JLinkedList class declaration

// Continued to next...
```





Example 19.2: Declaring a method to delete from END

```
// Continued on ...

// Defining the method to delete a node from the end
public T deleteEnd () {
    T x = null;
    Node temp = this.head.next, prev = null;
    if (temp != null) { // if the list is not empty
        while (temp != null) { // Move to the end node
            prev = temp;
            temp = temp.next;
        }
        x = temp.data;
        prev.next = null;
    }
    return x;
} // End of JLinkedList class declaration

// Continued to next... Null
```



Now, let us see the code for this method, this operation. We have to add one more method, the method is delete end and the body of the method is here which includes the necessary statements that is required. So here T is a temporary, x is a temporary, initially it is null. And, this basically starting from the header node. Now, if T is equals to null, it will take care about the empty list. If it is not null, then it will go, it will do these things. Now, here if it is not null, then this while loop basically moves from the current node, that is the header node, to the last node.

So, this is the usual, what is called the movement operation that you should do. And you can check that we are maintaining two pointer, previous and temp. Previous is basically one node ahead of the currently traversing node. And then when we come at the end, so we copy the value of the temp node into x and we make the null pointer of the previous node as a null, a link field of the previous node as null and that is all. So this is the method that we can follow in order to delete a node from end.

(Refer Slide Time: 09:57)



Deletion operations: Overview

- Deletion operations
 - Deletion at front


```
public T deleteFront() {  
    ...  
}
```
 - Deletion at end


```
public T deleteEnd() {  
    ...  
}
```
 - Deletion at any position


```
public void deleteKey(T key) {  
    ...  
}
```



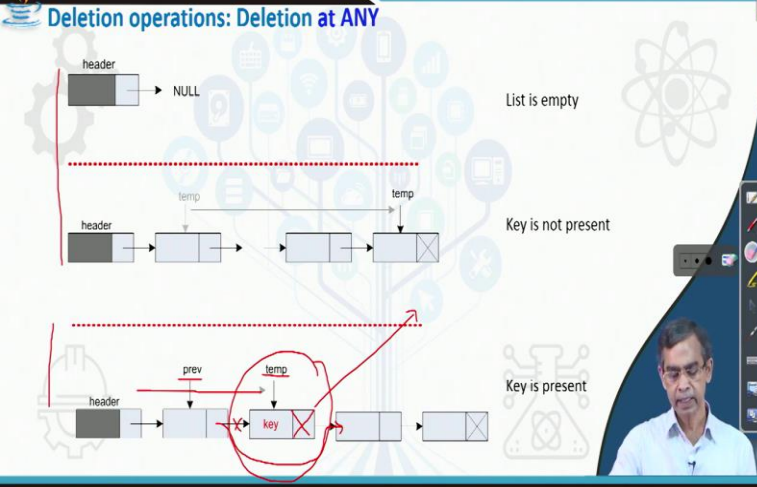


Deletion operations: Deletion at ANY

header → NULL → List is empty

header → [] → [] → [] → [] → Key is not present

header → [] → [key] → [] → [] → Key is present

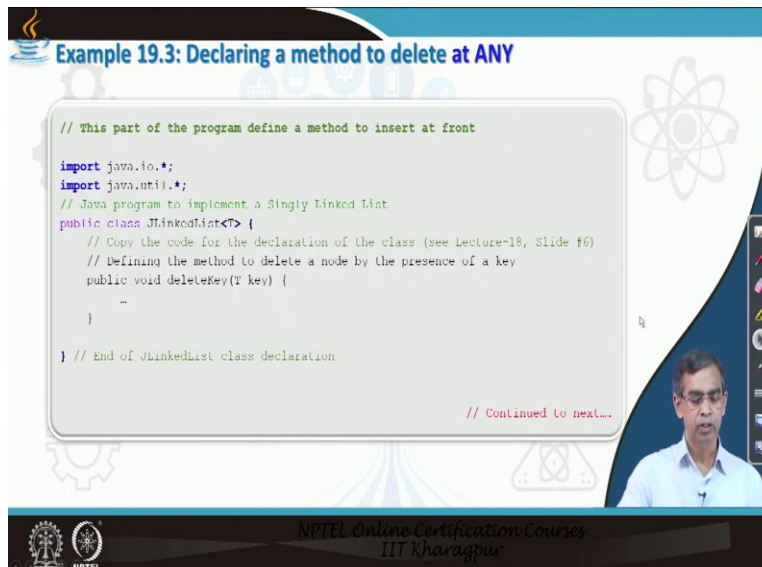


Now, our last operation, this is delete from any position, that position will be dictated by the value of the node which basically passed as an argument to this method. So, let key be the value. We want to delete a node having this value key. Now, again there, there are three situations we may have. One situation is that list is empty, then other situations is list is not empty but key is not present in the list, and then the third situation is that list is not empty and key is present.

Now, in the first two situations whether list is empty or list is not empty but key is not present, in that case we do not have to delete anything and that is all. However, if the list is not empty and key is present in the list, then what we have to do, starting from the header we have to traverse and reach to the node where the key value present. So, let temp is the node where it contains the key value, key. And while we are traversing to the target node, we have to keep a track of the previous node all the time.

So, when we reach it to this node, so let previous be the previous node. Now, once we reach to the target node, our link manipulation is like this, we have to assign this previous dot link field by the address of the node which basically points by the temp, that mean temp dot link. So previous dot link is equals to temp dot link and it will make this kind of pointing. Then (fin) we will do this null and finally this is already done. Okay? So, this is, this node is then free and we have to return this free node into the memory bank. So, this is the link manipulation that is required.

(Refer Slide Time: 12:21)



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

```
// This part of the program define a method to insert at front

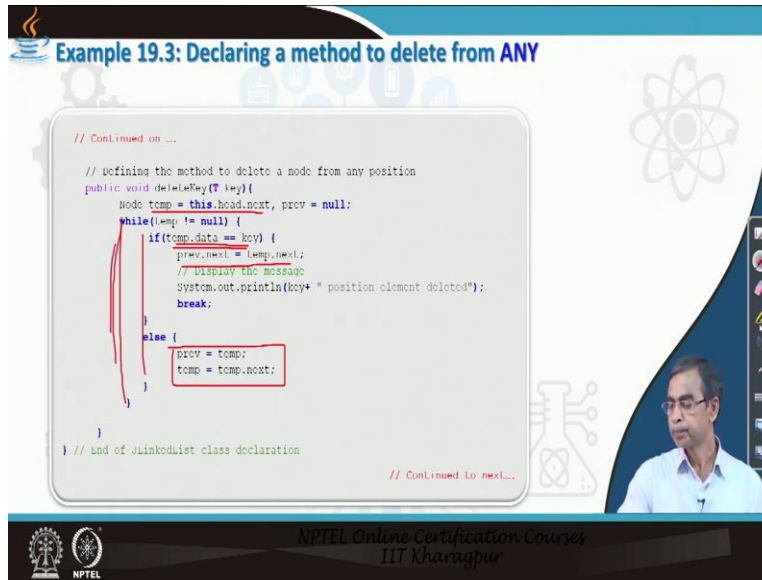
import java.io.*;
import java.util.*;
// Java program to implement a Singly Linked List
public class JLinkedList<T> {
    // Copy the code for the declaration of the class (see Lecture-18, Slide #6)
    // Defining the method to delete a node by the presence of a key
    public void deleteKey(T key) {
        ...
    }
} // End of JLinkedList class declaration

// Continued to next...
```

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

Example 19.3: Declaring a method to delete from ANY

```
// Continued on ...  
  
// defining the method to delete a node from any position  
public void deleteKey(T key){  
    Node temp = this.head.next, prev = null;  
    while(temp != null) {  
        if(temp.data == key) {  
            prev.next = temp.next;  
            // Display the message  
            System.out.println(key+ " position element deleted");  
            break;  
        }  
        else {  
            prev = temp;  
            temp = temp.next;  
        }  
    }  
}  
// end of JLinkedList class declaration  
  
// Continued to next...
```



Now let us come to the method declaration. So method declaration is there. Here like the other two methods for deletion, we want, we have to add one more method delete key. Here in this case, the method includes these statements, the statements are like this. We have to start with the header node. And here if you see, while temp not equals to null, if temp is not equals to null, we have to come to end, we do not have to do. The temp is null means we do not have any node there, list is empty.

Now if list is not empty, then while loop, we will enter into the while loop and here we have to, in this while loop again, we have to go to the target node. So there is while loop that temp, here we will see. We will just move one by one. Here is a movement. We will continue this movement until temp dot data equals to key. We do not write this, this is the target node actually. So we reach to, so if we continue this we will ultimately if the element is present there, we will be able to reach to the target node.

If we do not find at the end node, we will not do anything, so while loop will continue till. And then if we reach to the null, then we will not do anything. But if we find that key is present there, then we will just simply element is deleted. So this deletion will take place like this. This is the two, I mean link manipulation as you see it here. And this way it will continue the deletion. So, this is the case that you can follow, so for deleting a node from the list which is, which is, which contains the element and it is also not empty.

(Refer Slide Time: 14:08)

The slide displays a Java code snippet for a linked list. The code is as follows:

```
// Continued on ...
class LinkedListTest {
public static void main(String[] args) {
    LinkedList<Integer> list = new LinkedList<Integer>();
    list.insertFront();
    list.insertFront();
    list.insertFront();
    list.insertFront(4);
    list.insertFront(3);
    list.insertFront(2);
    list.insertFront(1);
    list.insertFront(0);
    // Print the LinkedList
    list.printList();

    list.deleteKey(3);
    list.printList();

    list.deleteFront();
    list.printList();

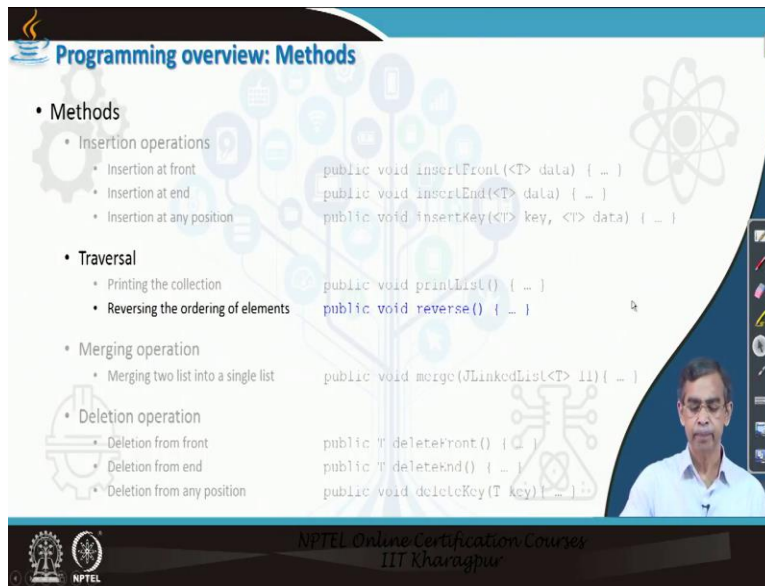
    list.deleteEnd();
    list.printList();
}
}
```

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

Okay, so we have discussed three different methods related to deletion operation. And here is the one driver method, the master program which basically check or to test how the operations works. Now here let us see, again we have to create a linked list. Let the name of the list be, list. And we just insert few nodes into the list. Here we insert 1 to 8 numbers. After the list is loaded, then we print the list. And then subsequently we follow certain (delete op) deletion operations.

Here delete key, that means it will delete this node, then delete front, delete end. You can call many other delete method here also. So, this way you can see at each point it will do the deletion operation and then it will, in this case, suppose you can call delete key 12. So, in that case also you will see because key is not present here, but it will not delete anything, because nothing to delete in that case, likewise. So you can test this program and then see whatever the method that you have defined, it works for you.

(Refer Slide Time: 15:35)



Programming overview: Methods

- **Methods**
 - Insertion operations
 - Insertion at front
 - Insertion at end
 - Insertion at any position
 - Traversal
 - Printing the collection
 - Reversing the ordering of elements
 - Merging operation
 - Merging two list into a single list
 - Deletion operation
 - Deletion from front
 - Deletion from end
 - Deletion from any position

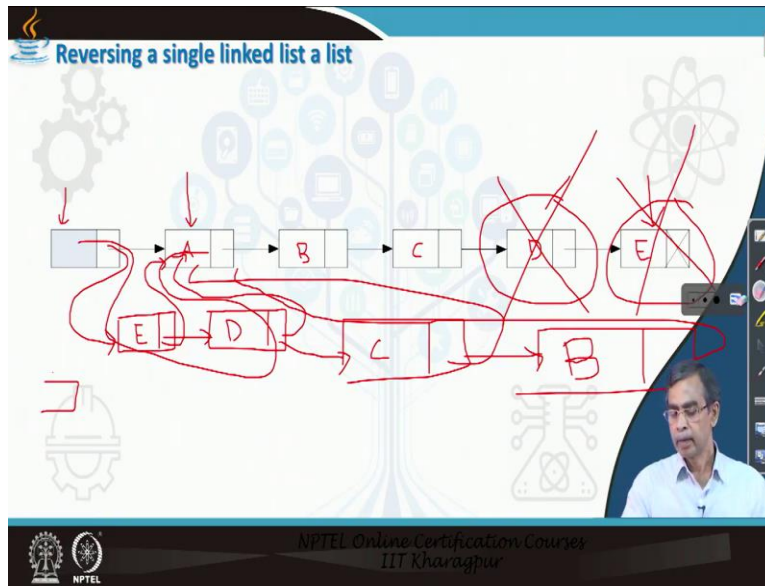
```
public void insertFront(<T> data) { ... }  
public void insertEnd(<T> data) { ... }  
public void insertKey(<T> key, <T> data) { ... }  
  
public void printList() { ... }  
public void reverse() { ... }  
  
public void merge(JLinkedList<T> ll) { ... }  
  
public T deleteFront() { ... }  
public T deleteEnd() { ... }  
public void deleteKey(T key) { ... }
```

NPTEL Online Certification Courses
IIT Kharagpur



Linked List Reversal

NPTEL Online Certification Courses
IIT Kharagpur



So, this is the deletion operations. Now, we shall learn about reversing a list. Now, list reversing is pretty simple actually. But you have to be little bit understand about how actually the reversion operation occurs here. Now, suppose this is the linked list and this linked list contains few data. Say this list contains A, this (list) this node contains B, C, D and E. Suppose this is the list that is given to you and this is the header. This is the header. Now, we have to reverse.

Reversing means that this list will contains, this node will E, then D, then C, then B and A. That means it is the reverse order, whatever the order it is given there in the reverse order. Now we can carry out this reversal operation. I am just okay, going to explain you the mechanism of reversing. There are many, many techniques. Many techniques are known, but we will discuss one simple most technique that is easy to understand.

So, the idea is basically the principle behind this technique is that, delete the last node that means end and insert the same node at front. If you do it, then your, I mean your reversing operation is (tak), is okay done. And at the, and then this method we have to repeat, delete at end and then the same node should be insert at front. We have to repeat till all nodes are basically for go through this operation. Now how it can be done? So initially, we are here and then we can come to the first node.

So this basically the, let this be the current node. And then starting from this current node, we will go to the last node. So this is the last node. Now so we know this last node, and then we, we

(should), we shall delete this node. So delete A, but this delete means we will just make a copy of this node actually. And then, that means copy of this means, the value of the node E is copied. So, then let the value of E is there.

Then what we should do is that, the current node which is here, just before this current node we, we have to, we have to insert the, this node. So this node, this node will be, so here. So this node we can get a node and we can make a copy of this node E. And then we can just insert it, insert at this point and this one. And then this basically delete. Okay, fine. So, this is done. Again, we have to repeat the same thing.

Again starting from this current node and then next is the last node is D, so we have to, we have to delete again this one. So in that case again, delete this last node and insert at the front. So our next time, so this will be here, sorry. We should delete just before to the current node. So this node will be deleted here. So it is like this, so D. And then we have to insert this D node just prior to this node, so we can make into this pointer and then this pointer we have to make to this one.

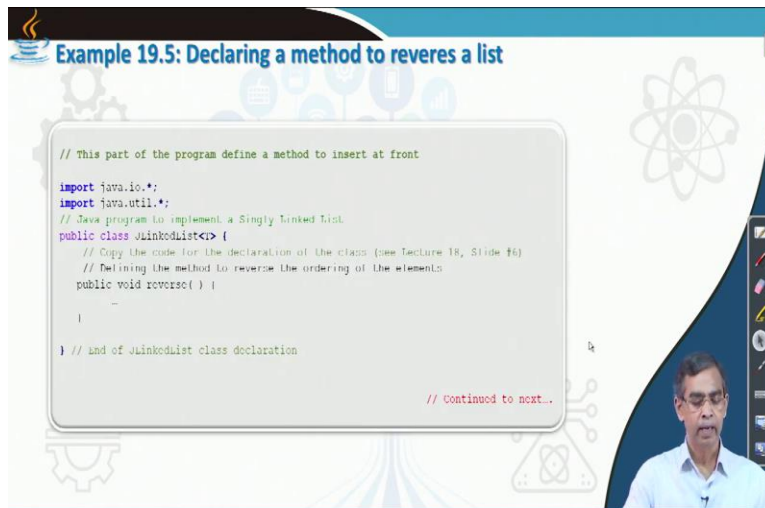
So this way you can see, this is basically deleted and this is become deleted. And if we continue this on, then C will be added here. So this will be there and it will be there. And then finally B, so it will be there A and here. And when there is no more the last node like, we can stop. So when we reached here so there is no more node to be deleted at the end, so we can stop. And this way we will be able to reverse it. So this is basically the mechanism and you can follow the mechanism actually again.

(Refer Slide Time: 20:48)

Example 19.5: Declaring a method to reverse a list

```
// This part of the program define a method to insert at front
import java.io.*;
import java.util.*;
// Java program to implement a Singly Linked List
public class JLinkedList<E> {
    // Copy the code for the declaration of the class (see Lecture 18, Slide #6)
    // Defining the method to reverse the ordering of the elements
    public void reverse() {
        -
    }
} // end of JLinkedList class declaration

// Continued to next..
```



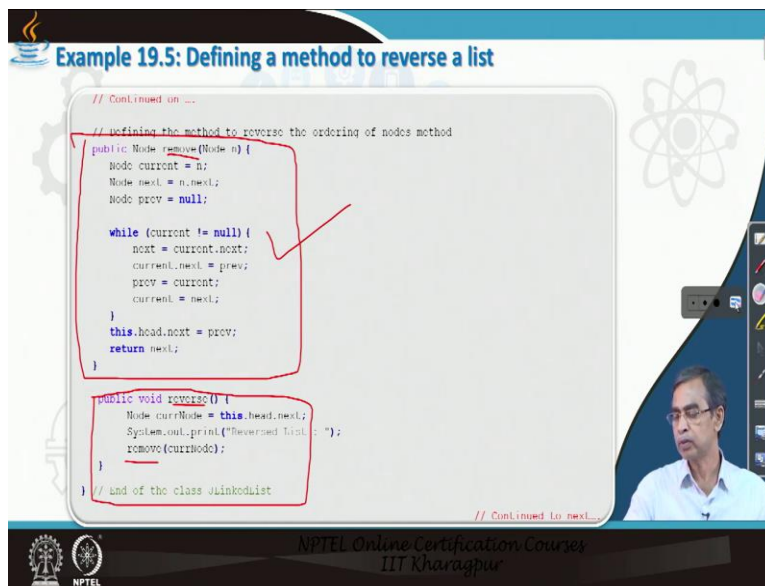
Example 19.5: Defining a method to reverse a list

```
// Continued on ...
// Defining the method to reverse the ordering of nodes method
public Node reverse(Node n) {
    Node current = n;
    Node next = n.next;
    Node prev = null;

    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    this.head.next = prev;
    return next;
}

public void reverse() {
    Node currNode = this.head.next;
    System.out.println("Reversed List : ");
    remove(currNode);
}
// end of the class JLinkedList

// Continued to next..
```



And let us see the code for this. I made a code for you. You can check this code and then you, you will be able to understand that how this code works for you. And so the reverse method operation regarding reversal and here is the code that is basically there, and this code is basically to go to the last node and (de), and delete that last node and then make a copy this last node just before the current node. So this copy, this method does there. And we have to just call this method, starting from the current node. This is basically the main operation of the reverse and this is basically sub-operations actually.

So that this reverse call this remove node, this remove node is defined here. And this will work for you. You can take this one and also you can find many other methods which is available in many textbooks, you can follow also and you can try to implement regarding the reversal. On many books, they recommend to perform this reverse operation using some recursive method. That is also interesting to note. I mean you can study it and you can implement using the standard procedure that I have discussed in this class.

(Refer Slide Time: 21:52)

```
// Continued on ...  
  
class LinkedListReversalDemo {  
    public static void main(String args[]) {  
        JLinkedList<Integer> list = new JLinkedList<Integer>();  
  
        list.insertEnd(9);  
        list.printList();  
        list.insertFront(5);  
        list.printList();  
        list.insertEnd(10);  
        list.printList();  
        list.insertKey(1, 5);  
        list.printList();  
        list.insertKey(12, 0);  
        list.printList();  
        list.insertKey(13, 10);  
        list.printList();  
        list.insertFront(3);  
        list.printList();  
  
        list.reverse();  
        list.printList();  
    }  
}
```

So this is the operations related to the single linked nodes. Now again utilizing the different methods related to insertion, deletion and then reversal, here is another master program that you can call different methods for a given list. So you can prepare a list using these procedures. It is mentioned there. Here only we have exercised the reverse method, the node that is being, those are being inserted into the list. Now if you call this reverse method, you will see they will (rev), they will reverse the ordering of the nodes.

So these are the different methods related to the linked list, programming linked list, single linked list rather. We have learned about it and it is just matter of practice. If you practice all programs one by one, adding one method each time and then testing the program, writing a master program you will be able to understand. And I should suggest you to practice it seriously.

(Refer Slide Time: 23:00)

The image shows two slides from an NPTEL presentation. The top slide is titled "Double Linked Lists" and features a coffee cup icon. The bottom slide is titled "Programming with double linked list" and shows a diagram of a double-linked list structure. The diagram illustrates a node with three fields: "LUNK" (left link), "DATA", and "RLINK" (right link). Below this, a sequence of nodes is shown, with arrows indicating the flow of data from one node to the next in both directions. The NPTEL logo and "NPTEL Online Certification Courses IIT Kharagpur" are visible at the bottom of both slides.

Now, let us come to the double linked list. Double linked list programming is same, same as single linked list. We have to first define a structure. In case of single linked list we have (create), we have defined structure with only one link. But, in case of double linked list we have to define two links in addition to the data, so just one more link. So you can replicate the program by modifying something which is required for double linked list.

So, you can define a node in case of double linked list and then linked list structure same as, they are also a header, here is also header. But header again, the header the left link of the header

should be made null, that consideration you should have. And then come to the different operations. All operations, if you follow one by one, then you have to see the link manipulation is not exactly same as the single linked list because for the single linked list only one link that needs to be manipulated.

Whereas for the double linked list, we have to manipulate two links for each node those are coming into the picture. So the link manipulation which we have discussed for the single, as for the during the discussion of this data structure and then algorithm that you should follow. And I kept this, I left this as an exercise for you so that you can practice of your own and then you can follow.

(Refer Slide Time: 24:40)



Otherwise, if you can follow the books here, the algorithms are there, that algorithm also you can follow. This algorithm is given in a pseudo code format. And you can follow this algorithm to implement all the operations related to the double linked list. And related to the application also in this book, you can, we have discussed lot many applications there. You can follow the different applications and then you can implement all those applications just writing the program.

And for each program implementation or application related, you do not have to define the same declarations again and again. What is the idea is that, you can maintain a package where your defined linked list structures and all methods in it can be there, and you can import this, this

package into your application program, and then just simply call the different methods whenever it is required.

For example, if you want to insert at end or delete from front, whatever it is there, you just call this method and it will work for you. So, regarding this application or writing a master program related to a particular application, you, it is left as an exercise for you. And all the programs which we have followed in these lectures and in previous lectures related to linked list programming is available in this link.

So and then in this link also include the detailed discussion about whatever the topics that we have discussed in these videos and in last few videos related to linked list structures. So you can check this link, you can get all the reference material there and then program including you can practice it. So, this is about the programming for linked list, second part. And, okay, that is fine. More practice that will help to, I mean, improve your skill. So that is okay, fine. I wish you all the best for programming endeavor. Thank you very much.