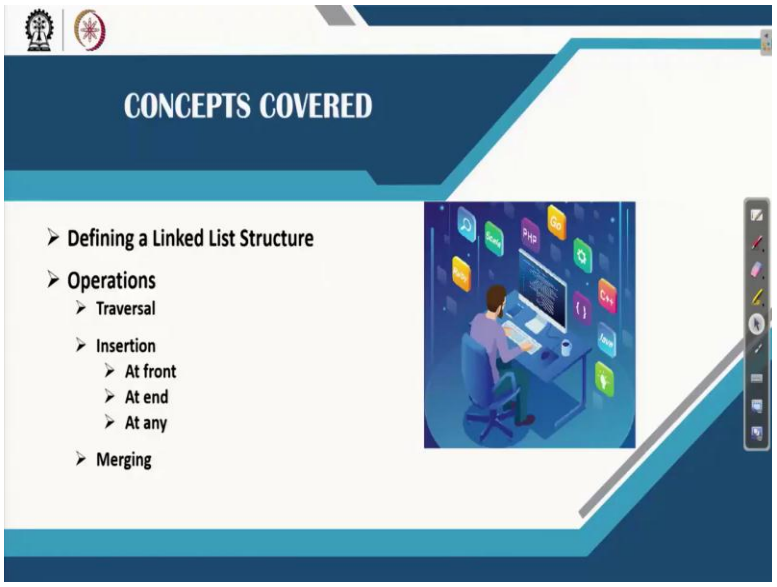
Data Structures and Algorithms using Java

Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Lecture No 18
Programming for Linked Lists (Part-I)

We have a basic understanding about linked list data structure. Today we will start about programming with this data structure, and for programming definitely we should consider the Java programming language. So today, linked list structure again have the different, you know type, say single linked list, double linked list, circular linked list.

(Refer Slide Time: 00:51)



CONCEPTS COVERED

- Defining a Linked List Structure
- Operations
  - Traversal
  - Insertion
    - At front
    - At end
    - At any
  - Merging

Mainly we will emphasize on in this lecture, emphasize on single linked list data structure. So, so far the programming is concerned so the idea is basically how the different operations on a linked list can be implemented.

Now again, I just before going to the actual discussion, I want to mention one thing is that, there are two way, the linked list can be used in any application development or any program; one using the custom concept. Custom concept means you can write the code and everything of your own. Another thing is you can use the package.

There is a package, already we have discussed about, that Collection Framework, Java Collection Framework that is defined in java dot util package. So this package provides you lot of functionalities to realize many data structure including linked list data structure. We will cover all aspects that mean programming of your own, also programming with Java

Collection Framework. But first we should start discussing about programming of our own that means how the data structures, that is the linked list data structures can be implemented in a program.

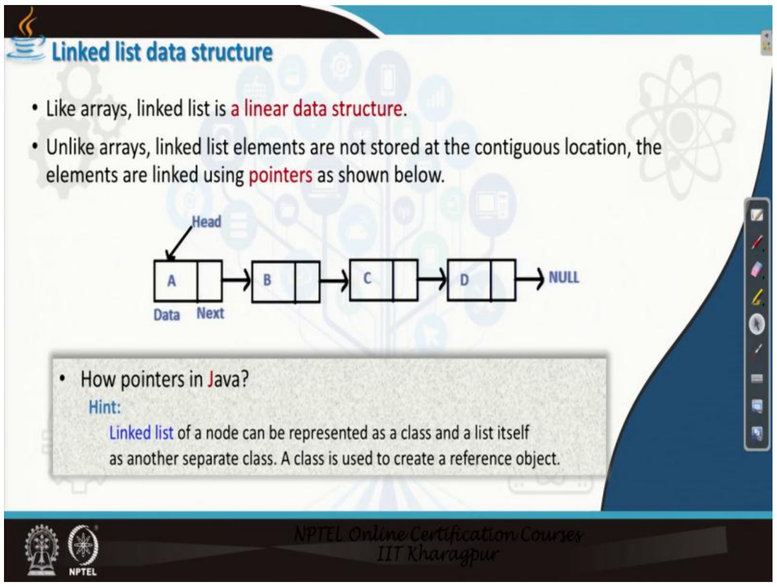
(Refer Slide Time: 02:30)

Linked List in Java: Defining

NPTEL Online Certification Courses
IIT Kharagpur
NPTEL

So today, there are two aspects actually so far the programming, I mean programming with linked list structure is concerned. First we have to define a list. So a list can be defined. Now this definition of list includes whatever the methods that we have discussed, namely insertion-related, deletion-related, then traversing, merging others can be defined. So linked list, programming for linked list is basically defining a linked list structure and then defining different methods of it.

(Refer Slide Time: 03:12)



Linked list data structure

- Like arrays, linked list is a linear data structure.
- Unlike arrays, linked list elements are not stored at the contiguous location, the elements are linked using pointers as shown below.

Head
A
B
C
D
NULL
Data
Next

- How pointers in Java?
  Hint:
  Linked list of a node can be represented as a class and a list itself as another separate class. A class is used to create a reference object.

We will discuss about these aspects, about linked list structure and then different operation on it. Now before going to discuss about linked list structure you can note that there are two things are to be defined properly. The first thing you have to define is, so for example this is a linked list. Now in this linked list, the list itself is a structure and you can note that this structure includes the different nodes in it. That means you have to define node and then using all nodes how you can define a linked list structure. So these are the two things.

Now again whenever we consider about defining a node there are again two things are to be considered. The first thing is the data and the next is field or link. Link basically, the node which will basically link to the next node. Now so far the data is concerned, the data can be of different types. So data can be integer, data can be string; data can be user-defined object like student or book.

Now here is again problem. So if you want to write a program that the linked list you want to maintain for integer or for string or for any other user-defined data type then you have to replicate the same program again and again but for different types. This is however is not a good, good way to solve the problem but we using Java and particularly using the generic concept that we have already learnt at the very beginning of this course, we can have only one realization, one implementation, only one programming. The same programming can take care any type of data. This is definitely is a very good, what is called the good point that the Java programming can provide you.

Second thing, second thing regarding the link so data can be of any type. Now link, now if we consider the link, link is what? Link is basically, is an address to the next node. Now if we see the address then address basically is a pointer. Actually the concept of pointer, because it basically point, so if we say this is the link. Link points to the next node. Actually this field store address of this node. So address of a node is basically pointer.

So what we can say in other word is that this field store the address of a node. So address means pointer, so this field we will store a pointer. Now again in Java there is no concept of pointer. Pointer is possible in C programming or C plus plus programming but Java there is no concept of pointer. So there is again question that how we can have the pointer business in this linked list implementation.

Java actually make the pointer concept very easy. If you define a, define an object say, of type of say, Student. So Student and then one object say, declare a X, so Student X semicolon. So that means you define an object, the name of the object is X and the object is of type Student. Now actually this X is a name of the object, essentially is a reference to an object.

What is a reference? Reference is basically a pointer to an object. So pointer means it is a memory location. So X essentially is basically memory location and it is a pointer. So this way Java can enable you the programming as easy as it is possible. And that is really a beautiful thing in programming because without any pointer headache that is there in C or C plus plus, you can have the same flavor here in Java.

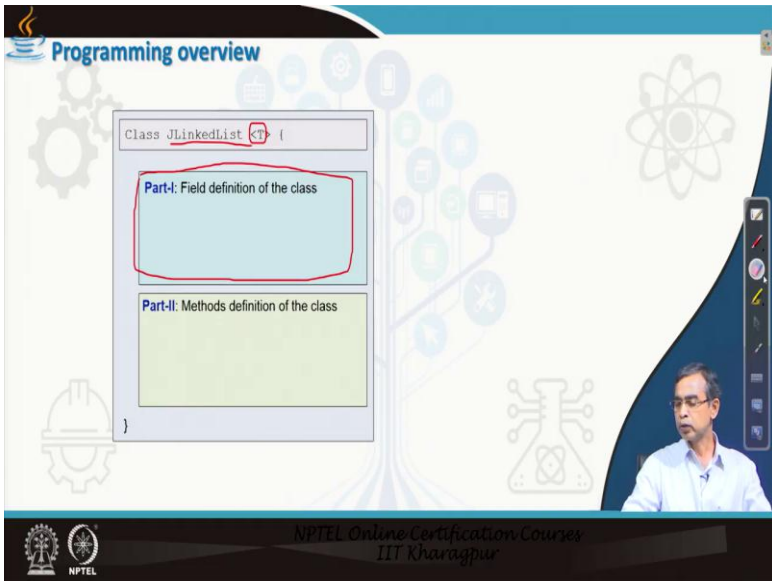
So we will see exactly how all those things can be done in our programming endeavor. My suggestion here before going to programming, I will try to give you the programming essence as much as possible. But you know understanding programming needs hands-on what is called the practice or rather we can say that you should try to understood the code, compile the code, run the program and then you can understand that what the program is doing.

So merely seeing the program you can understand only 1 percent of the programming what is called the benefits. But to have the best, what is called the skill, you definitely have to write the program and another suggestion is that while you are writing the program whatever you want to have certain modification, do it and see whether it is working or not.

At the worst case what it will do? Either compilation error or runtime error. Absolutely not an issue but by the process you can understand why this error is giving, why this error is coming, why this compilation is not successful, whatever it is there. But idea is that you should try to indulge yourself into the programming, and more programming practice will give you the more edge on the programming.

Anyway so I will give you all the programming ideas that we can think for linked list and you really enjoy that how such a nice things can be done using Java programming and particularly linked list is a very complex on concept, complex data structure which many programmers scared or fear that okay, oh, no, no we will not be able to do because so many pointers and everything, everything and usually end up with a incorrect result or either compilation error or so many things are there.

(Refer Slide Time: 09:56)



Programming overview

```
Class JLinkedList <T> {
```

Part-I: Field definition of the class

Part-II: Methods definition of the class

```
}
```

Anyway, so that is, now let us see how the programming can be done. I have already told you two aspects are there. You have to define a linked list. Defining a linked list means you have to define a node first and then the methods that basically require in order to manipulate the linked list and this basically following the logic that we have discussed in the last two video lectures in this module.

Now I can give you the framework of this programming. Systematically if we follow then we can understand a lot. Now first of all, we have to declare a generic class. Why we should declare a generic class? Because our intention that our linked list should store data of any type. Any type in the sense that it is integer, string, or whatever it is there. So we can say that it should store object of any type.

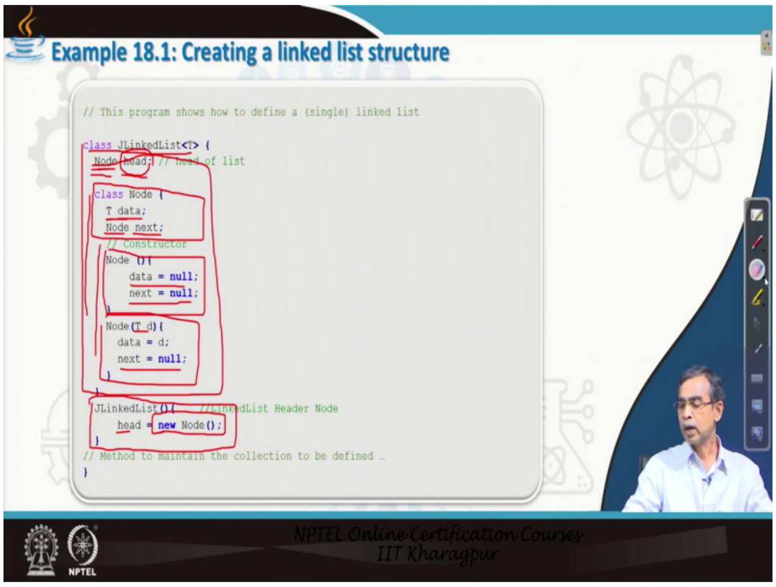
Now in order to declare a generic, as you already familiar to the generic programming we can declare a generic and this T is basically template that it can hold any type of data, the datatype T. T can be any type. Then the next part is basically declaring the structure of a node. That is basically you know, for example single linked list you have only one data part and then only one link. So that definition will be something. Then the double linked list or some other type of linked list.

So these basically includes how you want to define. For example, data can be again different way you can compose or whatever it is there. So all these things should be taken care in the first part. It is part 1. And then second part is basically different methods, different methods related to insertion, deletion, traversal, printing the list or merging whatever it is there. So the different methods can be declared here.

Now this way the entire class can be declared. This is the usual concept of Java programming, the class declaration or class definition which includes field declarations and the method declarations and method definition, body of the methods. So this is the basic idea about, that we should follow and let us see how actually our programming for the linked list can be carried out.

First we should discuss about how the field that means the node can be defined and then finally we will go for discussing about different methods one by one. As the number of contents are high, so we should discuss slowly and possibly in two lectures videos, the current one and the next one.

(Refer Slide Time: 12:36)



Example 18.1: Creating a linked list structure

```
// This program shows how to define a (single) linked list

class JLinkedList<T> {
  Node head; // head of list

  class Node {
    T data;
    Node next;
    // Constructor
    Node (){
        data = null;
        next = null;
    }
    Node(T d){
      data = d;
      next = null;
    }
  }
  JLinkedList(){      //LinkedList Header Node
      head = new Node();
  }
// Method to maintain the collection to be defined …
}
```

Now look at this program. This is a very simple program. You have to look it little bit carefully. I will try to give you the idea about how we have solved this problem. Now first of all, this is the usual concept that we are declaring a generic class and T is the generic facility that we can have any type.

Now here Node, we declaring Node because linked list consist of a node. So this basically we create the head. This is basically the field, we told that header. There should be a node called the header which is a dummy node actually which does not have any data and link. So this header is itself a node. So we define Node is a basically type.

We will discuss the Node type right now, so Node and header is also a Node. So basically linked list is a collection of nodes, collection of nodes of type this Node. Now let us define what exactly a node it is. So we have another inner class declaration.

In this inner class declaration, a Node is defined by two components, one is called the data and this data is of type T. It can be integer, it can be float, it can be double, it can be String, it can be Student, Book like this one. So we declare the T. So if like this. And then this basically is a link. Next is basically the link, link to a node. So that is why type is this one. So this completes the declaration of a Node actually.

This is very simple. So there are two parts, Node, Node is defined as a T of type data and then the next field. Now then in order to create the nodes, create nodes and then finally linked list we should have a constructor so that means how a node can be initialized or can be created. So here in this program we have defined two constructors, the first is a default constructor. As you see it is basically create a Node for which the data is also null, next is null, means both the fields are null value.

Then another constructor is basically take the argument data and it basically initialize the data field by this argument passed and then for the initial creation of the Node there is no link so we made next as null. So this is a two, these are the two constructors that can be created, that can be considered while you maintain a linked list. And then here you see, so this basically gives the declaration about the nodes in this linked list structure. Now here we want to create a linked list. So this is about Node.

Now what about the list? List is again another object. Now this object has, these are the fields that we have already declared. Now here comes to the linked list itself because how it can be created. So in order to create a linked list we again require one constructor. So here we declare a default constructor which is very simple in form. Here you see the default constructors and whenever we create a linked list, initially it contains only one node namely the header node.

So that is why we create head, head already declared as a Node type and new, so you see we create a default Node new, I mean using the default constructor Node new, that means header is initially created for which data is null and then link field is also null. So this way you can define the linked list structure, the linked list structure itself and then the node in it. So this basically the first part, part of the, part 1 which basically give you how a linked list can be defined.

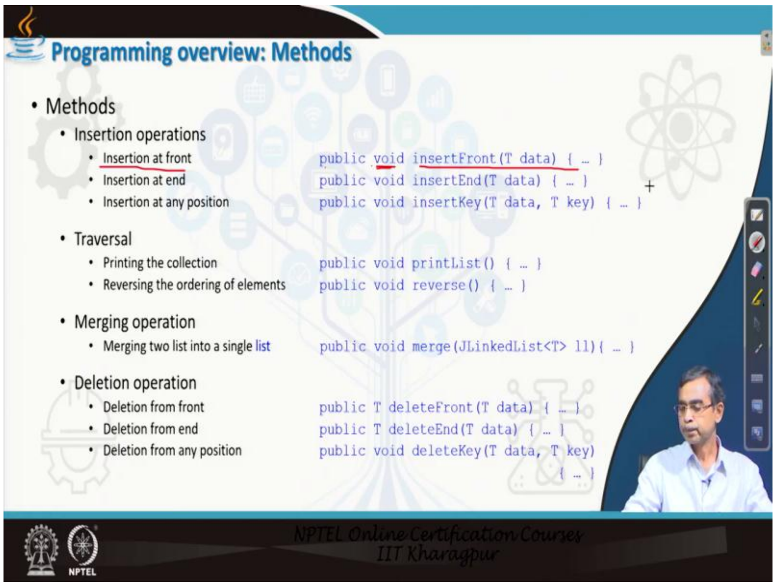
(Refer Slide Time: 16:48)

Methods for Linked List

NPTEL Online Certification Courses
IIT Kharagpur

Our next part is basically methods. So there are different methods can be defined, as many methods you can go, you can add into the, this class declaration, so like this one.

(Refer Slide Time: 17:00)



Programming overview: Methods

- Methods
  - Insertion operations
    - Insertion at front — `public void insertFront(T data) { … }`
    - Insertion at end — `public void insertEnd(T data) { … }`
    - Insertion at any position — `public void insertKey(T data, T key) { … }`
  - Traversal
    - Printing the collection — `public void printList() { … }`
    - Reversing the ordering of elements — `public void reverse() { … }`
  - Merging operation
    - Merging two list into a single list — `public void merge(JLinkedList<T> ll){ … }`
  - Deletion operation
    - Deletion from front — `public T deleteFront(T data) { … }`
    - Deletion from end — `public T deleteEnd(T data) { … }`
    - Deletion from any position — `public void deleteKey(T data, T key) { … }`

NPTEL Online Certification Courses
IIT Kharagpur

So here is idea about different methods that we can think about in this data structure based on our discussion that we had, we had in the last two lectures. So here we have listed the insertion-related operations, the traversals, merging and deletion operation. So we will discuss all operations one by one and here you can see insertion operations again can be based on different cases.

Why different cases, because for different situation the number of links that needs to be manipulated are not same. If it is at the front only fewer links, if it is at the end again fewer links but if any position then few more links like this one. So that is why three different cases that we had discussed in earlier.

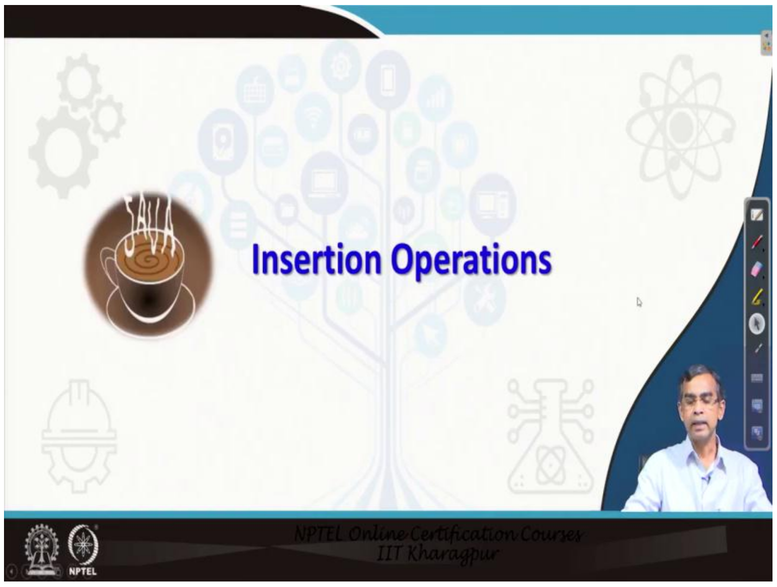
Now we will give you an idea about, in the programming so that if you little bit do the programming bit cleverly then you can do only one implementation but taking care of all cases. Definitely this is a clever process but what actually you should do, after implementation, after definition when you want to run the program you should test whether your program can satisfy all cases or not. So that is why we have little bit discussed in a structured way different cases of insertion, deletion or whatever it is there. Now let us first start with insertion operation in this direction.

Now before going to this each method, so every method should have its own form. It is called the signature. We should decide because program should be designed very nicely so that you can systematically write code so that is why we have discussed the different codes for that. Now so for the insertion at front we have declared this method. Now you see it is written, it is basically all methods are declared as public because we want to make the class declaration accessible to any program that is why. If you want to restrict then you can declare private, protected whatever it is there.

Now our objective, our intention is not to maintain any access restriction. So let us make it public. Then, so return type, you see some methods return void, some methods return of type whatever it is there so because of the logic it is there, insertion at front does not return anything so it is void and whenever you want to insert it so you have to pass which data we want to insert. So data is basically is the data of type so insertFront. Like this one the different method here we have discussed.

Now this is very important because we follow signatures what we basically planned or designed so that we have a very robust and then systematic code. Now so this is about the method signature that we want to implement one by one. First we will start with insertion at front, and then insert end, then insert key.

(Refer Slide Time: 20:27)

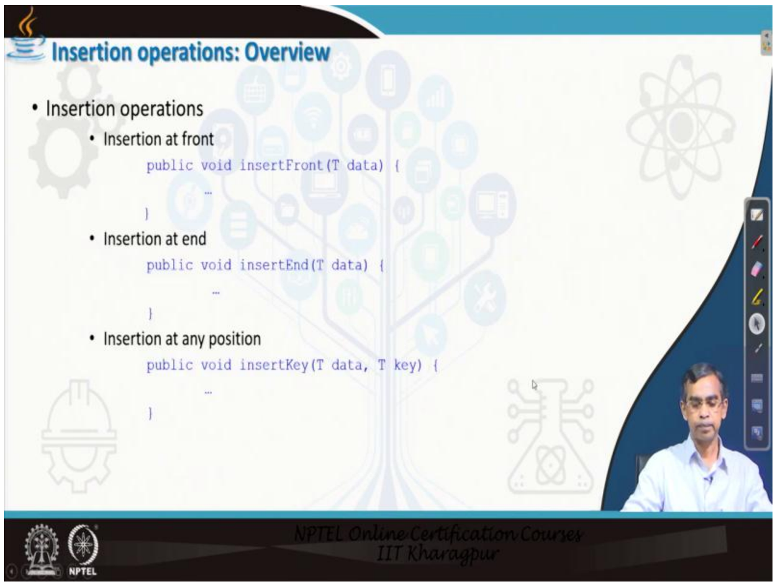


Insertion Operations

NPTEL Online Certification Courses
IIT Kharagpur

Now let us come to the insertion operation one by one. As we have already made the discussion is that different cases are there and we have to think about the different cases and how these different cases can be carried out. Let us first go, quick recapitulation of each cases and then the code for each method.

(Refer Slide Time: 20:45)



Insertion operations: Overview

- Insertion operations
  - Insertion at front

```
public void insertFront(T data) {
    …
}
```

  - Insertion at end

```
public void insertEnd(T data) {
    …
}
```

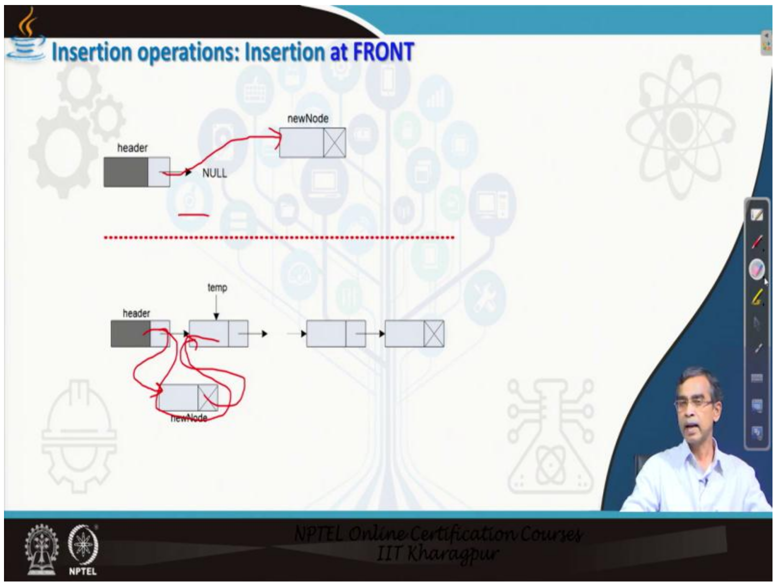
  - Insertion at any position

```
public void insertKey(T data, T key) {
    …
}
```

NPTEL Online Certification Courses
IIT Kharagpur

Now insertion at front three cases, or insertion operation having three cases, insertion, insertFront, insertEnd, insertKey.

(Refer Slide Time: 20:57)


Insertion operations: Insertion at FRONT

Let us first discuss about insertFront operation. So it is basically insertion at front. Now if we check it, insertion at front can face two situations. One situation is that we want to insert one new node to a list which does not have any node. It is initially empty. Now empty list means that it has only the header node.

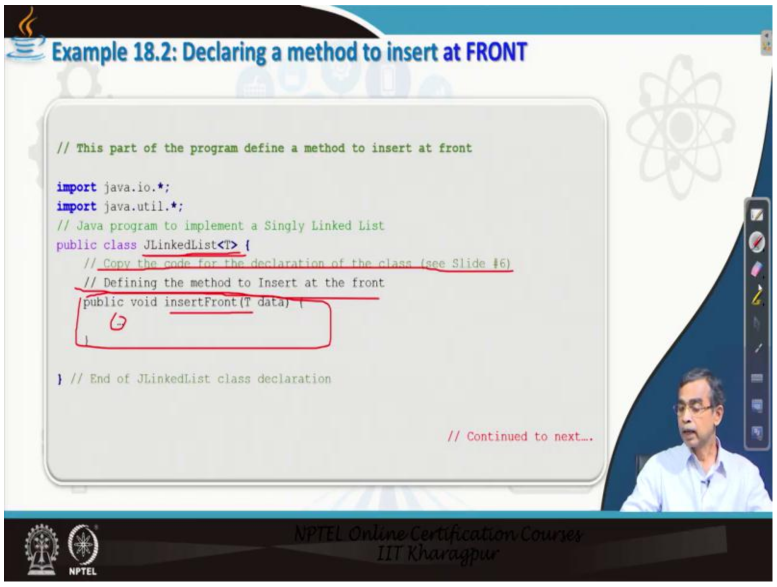
Otherwise we want to insert a node at the front in a list which contains at least one node in it, one or more node. So two situations, so these two situations are here. This is basically the first situation that the list is initially empty and the second situation is that the list contains at least one node.

Now, so this insertion if you see is a pretty simple. So this is the node that you have to allocate first and once the node is allocated, let the name of this node is newNode. This is the object so what we should do is that the value of this field next should be equals to newNode. So next, so header dot next equals to newNode basically makes the pointer to this one. So this insertion takes place.

Now on the other hand what the insertion here it is basically, this is a newNode and we have to make the first node which basically the header points so basically header points this temp. So newNode dot next is equal to temp if temp is the address of the next node so this basically make the pointer to this one so this pointer and then finally header dot next should be equals to newNode. So this way insertion at front will be taking place.

So now, this again, the operation should be done, this is the first one, this is the second one which we have already discussed in the last video lectures where we have discussed in details about this one. Anyway so this is idea about how the insertion at front can be done.

(Refer Slide Time: 23:19)



Example 18.2: Declaring a method to insert at FRONT

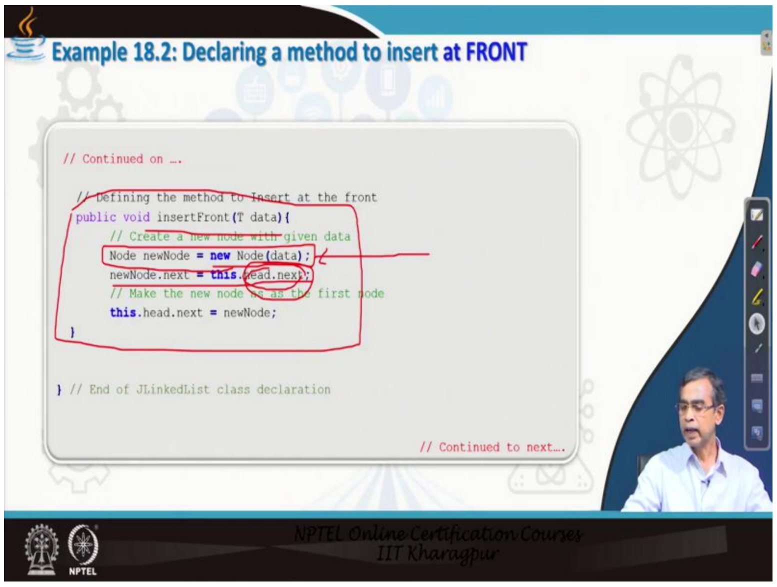
```
// This part of the program define a method to insert at front

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 Slide #6)
    // Defining the method to Insert at the front
    public void insertFront(T data) {
        ...
    }

} // End of JLinkedList class declaration

                                        // Continued to next….
```

Now let us have the code. The code I will explain you but you can run again the codes for your own understanding in your own space rather. Now let us have the code. How it will be? So these are the input is necessary because we want to include util packages, java dot io may not be necessary in this case particularly but in some situation it may require so you can, it is optional right now.

So we are declaring the class linked list so this is the name of our linked list class that we have already studied, discussed about that, different, okay the fields, copy for the code for the declaration of the class which we have already mentioned, discussed. And then the method, so here is the body of the method will look like this.

So public void insertFront T data and then body is basically we are now on the way to write code for this one. So this basically we are adding our first method to the linked list class declaration, the method is insertFront. The method has one argument of type T data. Now so, now let us see what is the body that is the main part of the method. Now body is basically considering two situations we have discussed. So let us see how the code will look like.

(Refer Slide Time: 24:41)



Example 18.2: Declaring a method to insert at FRONT

```
// Continued on ….

  // Defining the method to Insert at the front
  public void insertFront(T data){
       // Create a new node with given data
       Node newNode = new Node(data);
       newNode.next = this.head.next;
       // Make the new node as as the first node
       this.head.next = newNode;
  }


} // End of JLinkedList class declaration

                                        // Continued to next….
```

Now here is the code you see. So this is the body of the insert at front method. This is the front, now you see node, newNode, this basically allocate a node new that you want to insert into this one. So this is a first is called the allocation. So allocation means it will get the node from the memory bank.

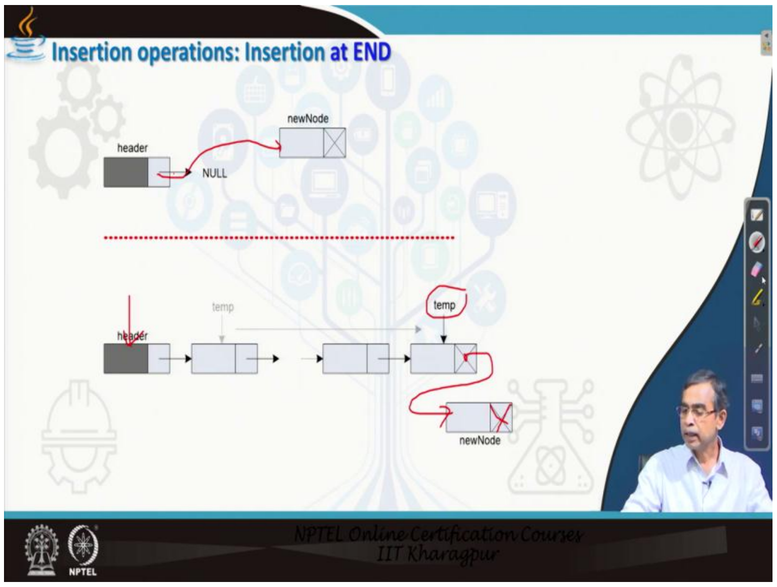
Now new operator will basically allocate the memory for you. Now newNode dot next, if you write this code what it does mean? Now head dot next is basically the address of the first, first node. And this basically is the linked list which we are considering that is why. This node's head, header so this basically newNode dot next is basically store the address of the first node in the list. Now, okay fine and then we make the header, link of the header node as the address of the newNode, so this one. So the two link operation is there.

Now here you can understand that this will work if the list is initially empty or not even, in the both cases will work for you, because if see suppose the list is empty, if it is empty then head dot next is basically null. So newNode dot next is equals to null. That means this is the last node. And then the head dot next will point to the newNode. So it will work.

Now again this will work obviously if the list contains at least one node, there also. So both way you see this code is written in that way so that it basically works for you for satisfying the both cases, both situation, either the list is empty or list contains at least one node in it. So this is the beauty that okay, little bit cleverly we have managed it so that we can, many cases, whatever it is there but compact way it can be. Otherwise you can have more if statement, more case, I mean check statement and then you can write it which is unnecessarily code is too lengthy but this is a very compact code that you can have it.

Little bit, at the first learning you may find little bit difficult but whenever you are habituated this is the first time you are handling with pointers in Java so you may little bit feel little bit difficult to understand but if you go on practicing and then you will be able to understand it very easily. I hope you have understood. Anyway in case of your difficulties and everything definitely you should pose your question to the discussion forum. Now let us come to the next operations.

(Refer Slide Time: 27:40)



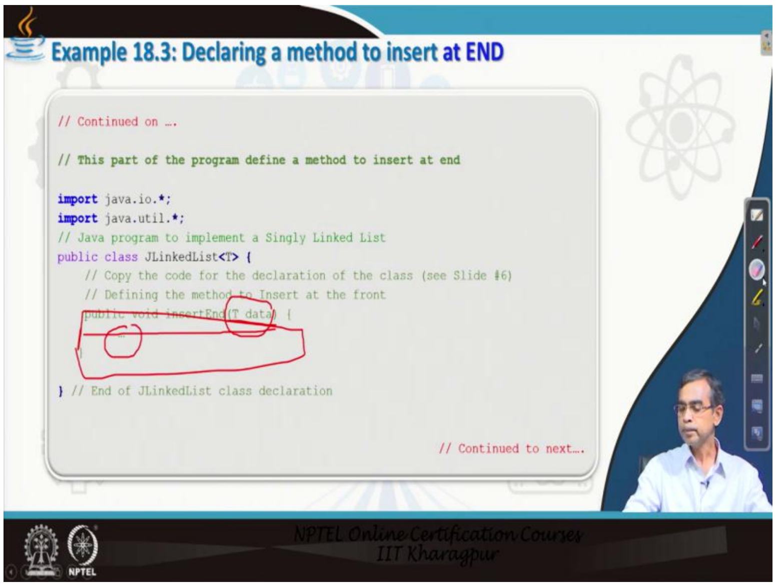
Insertion operations: Insertion at END

This operation is insert at end. Again like insert at front it has the two situation, the list is empty, there is no node in it. Or the list contains one or more node. Now whatever be the situation the idea will be like this. So if the list is empty and if you want to insert at end, the same as this field should be equals to this one. So that means header dot next is equals to newNode. That is all. Or if the list contains at least one element in it, so you should start with header because this is the only pivot from where you have to start. So header and then you have to traverse the entire list, go to the last node.

What is the last node? Last node is that node for which this next field is empty. So you have to just have a loop, traversing each node until you can come to a node for which the next field is empty. And let this address of this node and let the name of the address be temp. Once you know this, then adding this newNode into this list is very easy. Just simply make this next field is equal to newNode.

So this way it will add the node at the end. And definitely you should not forget about making the null field of this newNode as null. But whenever you create a new node by new operator it automatically create a node for you for which the data can be using the constructor, data can be initialized by the value that you passed and then this one. Now let us come to the details of the programming having this concept. For this the code is very similar.

(Refer Slide Time: 29:37)

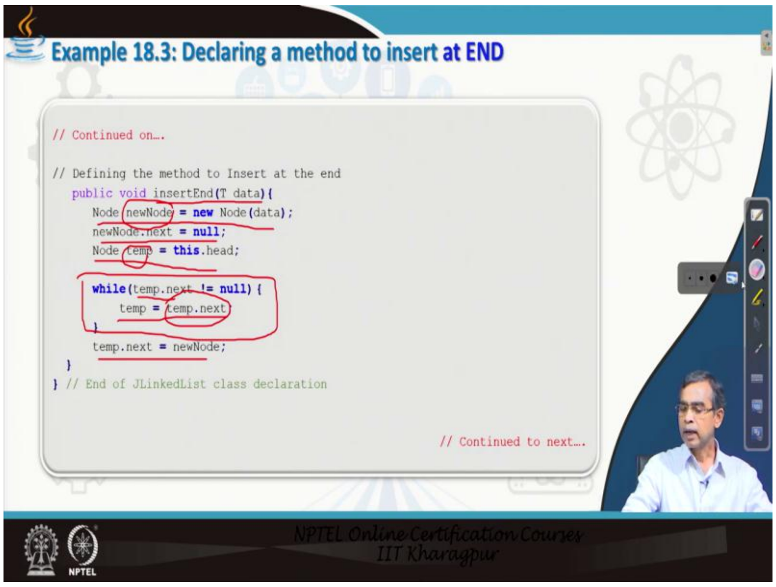


Example 18.3: Declaring a method to insert at END

```
// Continued on ….

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

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 Slide #6)
    // Defining the method to Insert at the front
    public void insertEnd(T data) {
        …
    }

} // End of JLinkedList class declaration

                                              // Continued to next….
```

What exactly the thing is, you have to add one more method. The name of the method as you have mentioned insertEnd. In order to add a, well data of type T, and this basically the method body and body is to be defined. Now let us define the body how it should be.

(Refer Slide Time: 30:07)



Example 18.3: Declaring a method to insert at END

```
// Continued on….

// Defining the method to Insert at the end
   public void insertEnd(T data){
       Node newNode = new Node(data);
       newNode.next = null;
       Node temp = this.head;

       while(temp.next != null) {
           temp = temp.next;
       }
       temp.next = newNode;
   }
} // End of JLinkedList class declaration

                                              // Continued to next….
```

Now here is the body of the method. The name of the method is insertEnd. Again two situations, empty or non-empty but we can write a compact code the way we have done for the insert at front, same way and code you can check how it work. So first we have to allocate a new node for the data that you have to passed. So newNode, so newNode is the name of the node that we are going to add into the list.

And this is okay, fine whenever you use this constructor, if you see the constructor automatically creates the null, link field null but you can do it also forcefully that is what we have made that, new node dot next is equal to null. That means this is a node whose link field is null. Then we create a temporary node, node temp this dot head initially start. Because temp is basically a pointer which basically used to traverse the entire list starting with the head node.

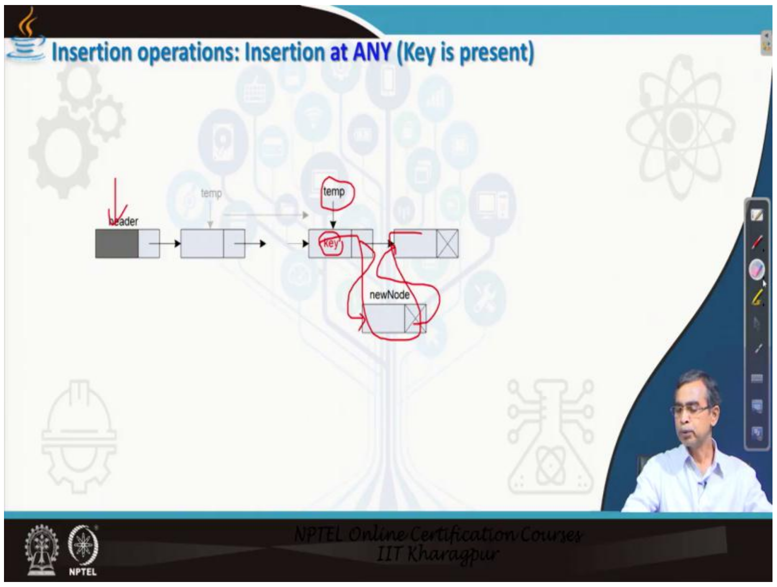
So this basically, this is the node and the linked list for which the header is considering. So this dot head is basically header of the list, and temp is basically point to this header. So, so this is the pointer. Now here this code you can understand what this code will do for you. So temp dot next that means we are checking whether we are at the end node or not. So if temp dot next is equals to null that means this is the node whose link field is null. Then we can say that it is the end fault. If it is not then we just simply move pointer from one node to the next node.

So how you can move it? This is the way of moving from one node to this one. So just changing the pointer from the current node to the node which basically store its address into the next field. So temp dot next is basically temp that means the pointer will move from one node to the next node. So this is the concept that you can follow very easily. And once you reach to the end node then you just simply make that link of this last node as a newNode.

Now again you can check whether two situations, when the list is empty it is working or not. So this code definitely it work for when the list is non-empty. Now if the list is empty then what will happen? So this is fine, no problem we have created a newNode and then we are presently at the header node.

Now whenever the list is empty then the pointer that it store, the header node is null. So this loop will not work because this will not rule. So you will directly come to here. What is the directly means this is temp is say presently the header node. So header dot next equals to newNode. That mean we just assign the link field of the header node by the address of the newNode, and that is all. So this is the idea about that how the two situations can be accomplished in one way of coding, no need of special checking whether this case or that case or like this one. So this is about insertion at end.

(Refer Slide Time: 33:31)



Insertion operations: Insertion at ANY (Key is present)

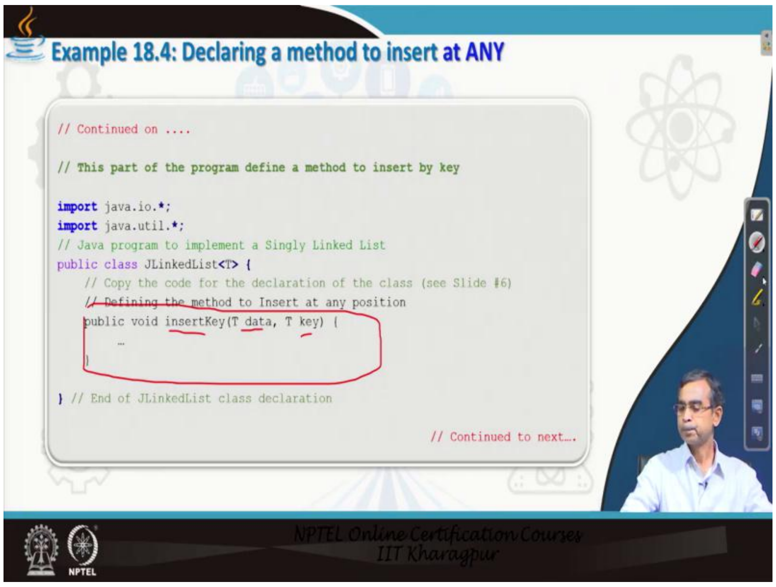
Now insertion at any. So in order to insertion operation whenever at any is there, there is basically, at any means basically we have to check that we want to insert a new node after a node whose value is a given value. So this given value is called the key value. Now there are again two situations for this.

First is that, key is not present in the list at all and that list can be again initially empty or it can contains one or more node, whatever be the situation but key is not present there. And the second situation is that key is present there, key is present there means definitely it is not that a list is empty. List is non-empty and it can contains one or more node. So this is the concept actually.

Now let us see how we can write the method for insertion at any considering whether key present or not, it is there. If key is not present then insertion will not do anything. So it is just simply nothing. But if key is present a new node will be inserted just after the node whose value, data value is key. So this is idea about, here is the pictorial concept that you can see. We have to come to the, so now starting from header you have to first go searching that whether there is a node which contains key or not. If it contains then which is the node actually it is there. So we have to come here.

So temp is basically is the node which includes the data value, that data value is key in this case. And a newNode we have to get it from the memory bank and then node needs to be inserted here. So insertion is like this. We have to set this pointer that means which basically points, assign this one and this pointer will be there. So two links are to be managed and this way a newNode will be added just after the key node. So this concept can be implemented writing few lines of code and now you will see exactly how the codes will be there.

(Refer Slide Time: 35:46)

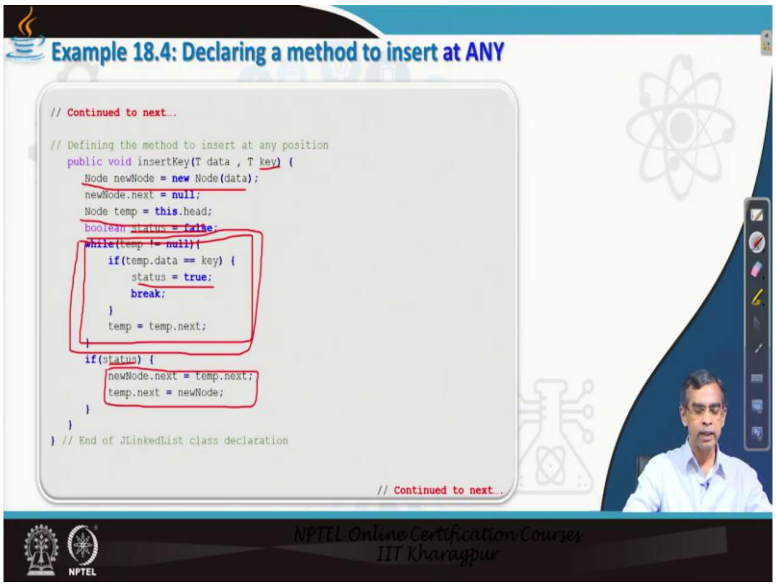


Example 18.4: Declaring a method to insert at ANY

```
// Continued on ....

// This part of the program define a method to insert by key

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 Slide #6)
    // Defining the method to Insert at any position
    public void insertKey(T data, T key) {
        ...
    }

} // End of JLinkedList class declaration

                                              // Continued to next….
```

So this basically, idea is that we want to declare the method, the next method is basically insertKey. For the insertKey this one key that mean this is the value after which new value, that is a new value needs to be inserted.

(Refer Slide Time: 36:11)



Example 18.4: Declaring a method to insert at ANY

```
// Continued to next….

// Defining the method to insert at any position
   public void insertKey(T data , T key) {
      Node newNode = new Node(data);
      newNode.next = null;
      Node temp = this.head;
      boolean status = false;
      while(temp != null){
          if(temp.data == key) {
              status = true;
              break;
          }
          temp = temp.next;
      }
      if(status) {
          newNode.next = temp.next;
          temp.next = newNode;
      }
   }
} // End of JLinkedList class declaration

                                        // Continued to next….
```

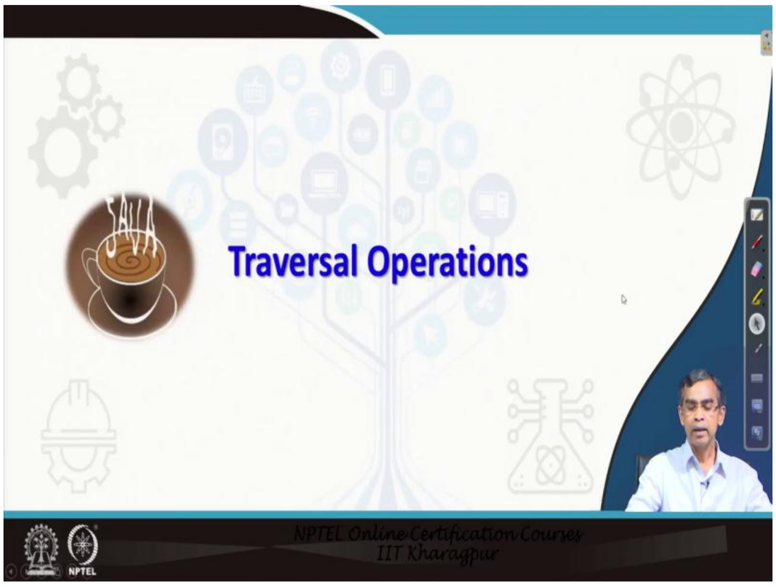
Now let us see the code. The code is again pretty simple like the previous codes actually. I will just explain it, this method that we are going to have the body now. So this basically get the newNode, then we initialize that, we have to start from the header and then status is the one thing that whether we can find node is exist or not.

So now in order to set whether the, already node is having which value is basically key value so this basically, a search is taking place and the search will basically, whenever it finds the key value is present, status will be made true and then it will break from the loop, because once you find it you no more need to traverse again. So stop there and come here and if you find the key value it is there so temp equals to temp next and even if, so this basically is the code by which you can make the traversal starting from the header node to reach to the node if it is there or not.

Now if key value is not there then status will be false. So if status is false, do not do anything. That means key does not present there. So we have do not have to; do nothing, anything. If key is present there, so that is why if status is true, now you see the node, that link management that I have mentioned two link, so newNode next will be the temp node, temp dot next; temp is basically where the key is present and then temp dot next becomes the newNode that it is there.

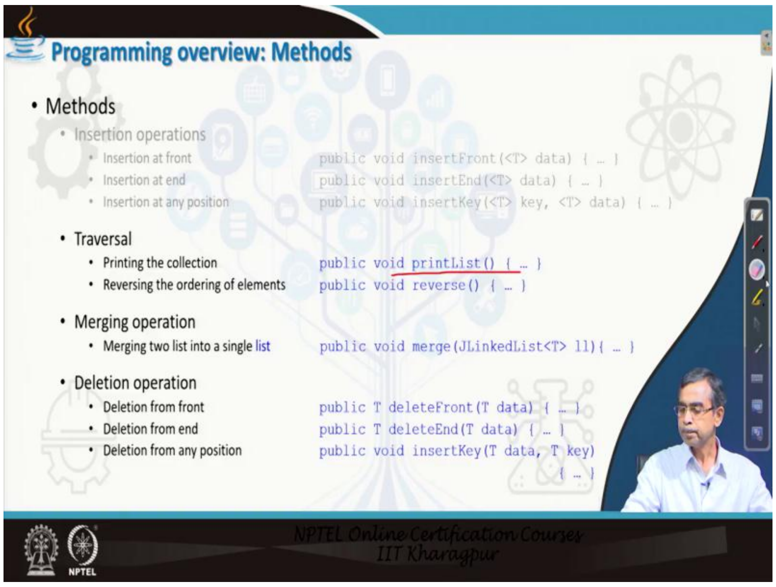
So these are the two pointers or link, link to be manipulated that is all. So this way insertion at any position can be carried out. Now you can again check the program, run program and check whether it is working or not. I will give you a simple master program one, so that you can check whether all methods that we have discussed, whether it is working or not.

(Refer Slide Time: 38:13)



Traversal Operations

NPTEL Online Certification Courses
IIT Kharagpur



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

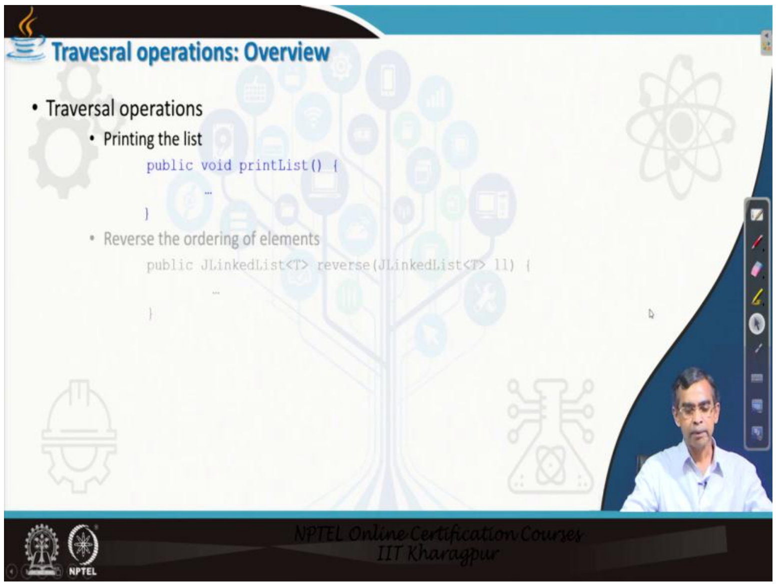
```
public void insertFront(<T> data) { … }
public void insertEnd(<T> data) { … }
public void insertKey(<T> key, <T> data) { … }

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

public void merge(JLinkedList<T> ll){ … }

public T deleteFront(T data) { … }
public T deleteEnd(T data) { … }
public void insertKey(T data, T key)
                                    { … }
```

NPTEL Online Certification Courses
IIT Kharagpur

So this is the, okay, this is, now let us, first going to the master program again. I just want to have one more method, I want to include it. It is called the printList. printList means it basically prints the entire list. A collection can be printed, of course using System dot out dot println by giving the name of the collection. Now in this case you see a linked list is basically a collection.

Collection means it is a collection of elements where a node, a node in this collection or in this linked list stores an element. Now let us see, first define a printList method going to a master program. Master program basically call the different operations for a given linked list, how a linked list can be created etcetera. Now let us come to the traversal operation first so that we can have the full idea about it.

(Refer Slide Time: 39:10)



Travesral operations: Overview

- Traversal operations
  - Printing the list

```
public void printList() {
    …
}
```

  - Reverse the ordering of elements

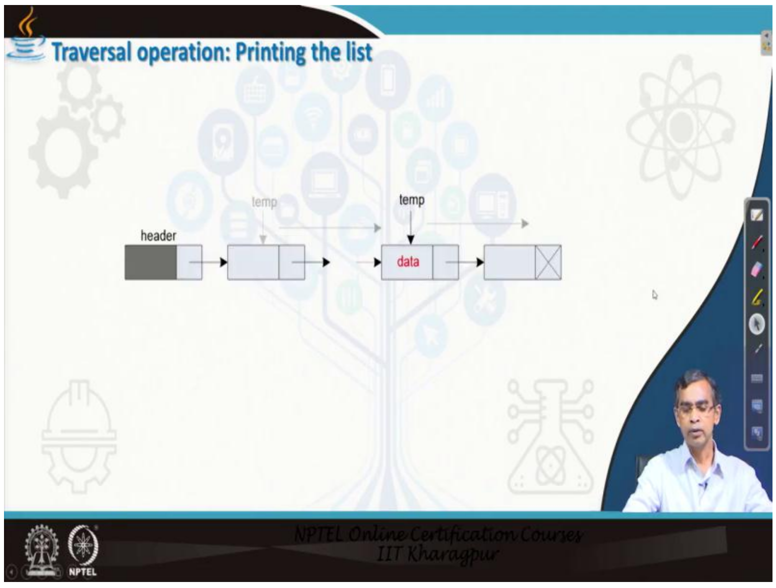
```
public JLinkedList<T> reverse(JLinkedList<T> ll) {
    …
}
```

NPTEL Online Certification Courses
IIT Kharagpur

So it is a basically traversal operation that we are going to plan is basically to print every node in the list. So that is why we have given the name is the printList is the method as it does not require any value to be passed so its argument is null and it should not return anything so that is why return type is void. We make the method public because any program can use it.

Now, so now printList is basically traversing. Traversing means it is just starting from the header node, you have to go to the next node, take, print the value that the node contains then move to the next node and you should continue this till you reach to the end node, end node means the node whose next link field is null. That is the idea actually, very simple idea.

(Refer Slide Time: 40:06)

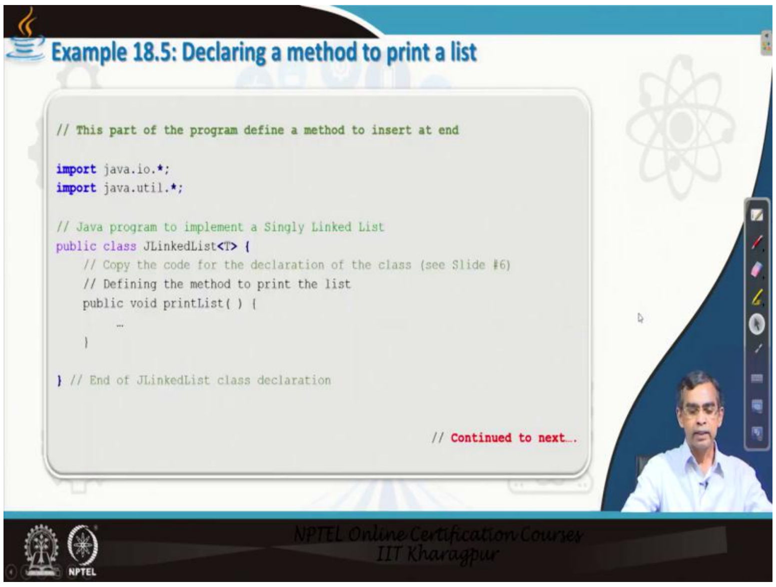


Traversal operation: Printing the list

NPTEL Online Certification Courses
IIT Kharagpur

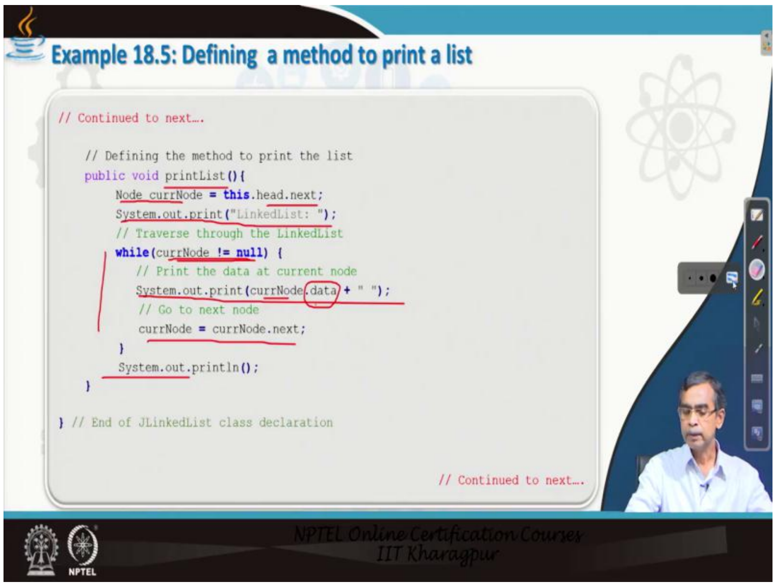
Now let us see how the program for this will look like this. It is basically simply a traversing, moving the pointer and how the pointer can be moved from the node to the next node? It is basically temp equals to temp dot next. This is a very simple procedure; temp is this one so temp dot next means this one. So temp, new temp will be there, that concept it is.

(Refer Slide Time: 40:25)



Example 18.5: Declaring a method to print a list

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

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 Slide #6)
    // Defining the method to print the list
    public void printList( ) {
        …
    }

} // End of JLinkedList class declaration

                                                // Continued to next….
```



Example 18.5: Defining a method to print a list

```
// Continued to next….

    // Defining the method to print the list
    public void printList(){
        Node currNode = this.head.next;
        System.out.print("LinkedList: ");
        // Traverse through the LinkedList
        while(currNode != null) {
           // Print the data at current node
           System.out.print(currNode.data + " ");
           // Go to next node
           currNode = currNode.next;
        }
        System.out.println();
    }

} // End of JLinkedList class declaration

                                                // Continued to next….
```

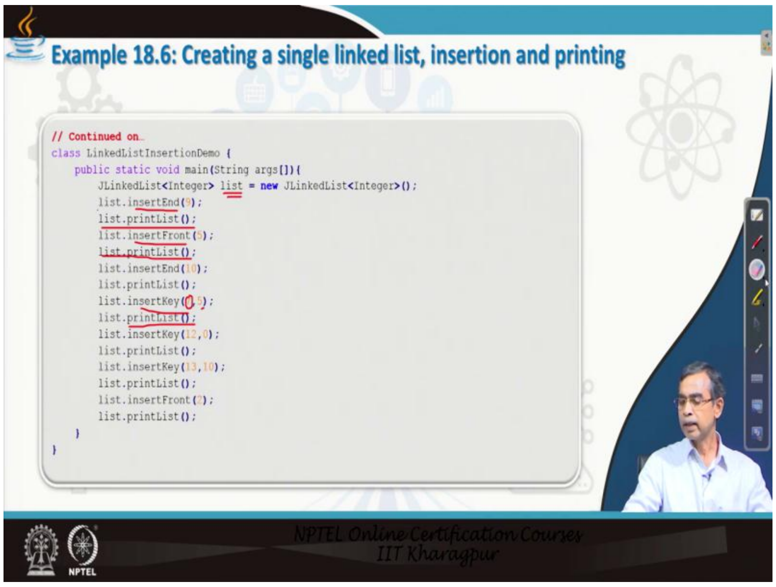
Now this is the code that we are going to write and here is the code you can check it, very simple code; so this is the name of the method. Now we start as a currNode. currNode is basically head dot next, head dot next means the first node. Now if it is empty then you stop it there. Now here you see System dot out println linked list, we are going to print the linked list actual, this is a simple message print.

While current Node not equals to null, if current Node equals to null that means if the list is empty it will directly come here, nothing will print. List is empty, no need to print anything. But if list is not empty, not empty then it will come here, System dot out dot println currNode dot data so whatever the value that the currNode contains.

So data field contains whatever the value for the current Node it will print. It can store any value, string, integer, double, or even the Student type where the name, roll number, marks whatever it is there, because your System dot out print method is very powerful. It can print anything and anything as the form of a string actually.

So this way actually you should not bother about any value can be your node can, your linked list can contain any type of value, it can print it. And then once you print the current Node you move to the next node. So next node means where the currNode equals to currNode dot next, so move to the next node. So this way you can traverse from the header node to printing each node in it and then displaying. So this is idea about the print method.

(Refer Slide Time: 42:08)



Example 18.6: Creating a single linked list, insertion and printing

```
// Continued on…
class LinkedListInsertionDemo {
    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(7,5);
        list.printList();
        list.insertKey(12,0);
        list.printList();
        list.insertKey(13,10);
        list.printList();
        list.insertFront(2);
        list.printList();
    }
}
```

NPTEL Online Certification Courses
IIT Kharagpur

And so once we are having the insertion operations and print method we are in a little bit position to have a driver program or master program. So this is the example of a master program. The master program basically create a linked list. According to our program implementation that we have done, we are now in a good position to write, create a list and then we can insert element according to our wish, insert at front or insert at any or insert at end whatever it is there and then finally printing the list.

So for which what we do, let us check it carefully. So we are declaring a class. Our program is a master program or driver program which include the main method and you can see in the earlier declaration there is no main method. Only you have, we are busy to define our class, the name of the class is JLinkedList, is a generic class and this class should contains a number of node and how insertion, how printing, traversing can be done.

Now, so we first create a linked list of type JLinkedList. JLinkedList our class declaration, now here you see generic, we want to create a list of integer. You, one thing you should note, you should not write int as a primitive data type. You should declare as a, is a Object type, okay. So that is why capital Integer. That mean we want to store integer value but it is an integer as an object we want to store.

Now this is the name of our list we get. You can give l, you can give x, no problem. And this basically create, initially create the node and for which we call the constructor that is there. So this constructor we will create a list which is initially empty. So this basically a declaration about how a node, how a linked list can be created and initially linked list is empty.

Once the linked list is created then we can go on adding node one after another or whatever it is. So here you see we can call insertEnd method for this list and argument that we pass 9. You can understand what it does means, it means that we want to insert a value, the value is 9, insert at the end. As the list is empty, absolutely no problem, it will insert anyway then insert at front.

Now okay, then print list so it will print the 9 value only. Next insertFront, now what you can see is that 9 is there, we want to insert 5 before 9. So you can do it and then print, you can, it will print 5 and 9. And so on, you can see the different insert operation can be insertEnd; insertKey that means here you see, we want to insert 7, this is a data, after a node which it contains 5. So basically the 7 will be added there in the node after 5. So 9, 5, 10; 9, 5 then 7, 10 so this will be the list, if you print list, call the printList method it will print 9, 5, 7, 10 like this one.

Now so on, we can go on printing, doing, so you can just, okay call the method as many time as you wish to increase the list according to this way. So this way you can learn about whatever the code you have developed so far, you are now very comfortably create a list of your own. And the list is basically, will be given or called by a name and this name as you have given, it is programmatic given name, the list you can give any other name also there and this list basically maintain the header and everything inside this one. So it is just like a linked list itself usually it looks like. So this is the program so far that we have discussed about, insertion operation into a linked list and different cases of insertion operation.

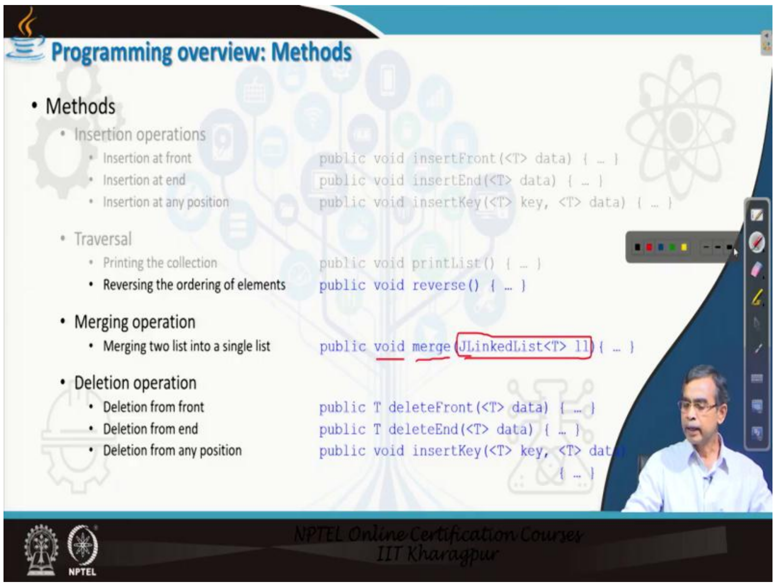
(Refer Slide Time: 46:28)

Linked List Merging

NPTEL Online Certification Courses
IIT Kharagpur

Now before going to just wrap up these things let us see the merging operation. So you know exactly what is a merging operation? It basically append one list after the another list. So there will be a master list and then this master list basically should call this method merging and then as a method it should pass an argument that which list you want to merge into this list. So now merge method needs to be defined.

(Refer Slide Time: 47:05)



Programming overview: Methods

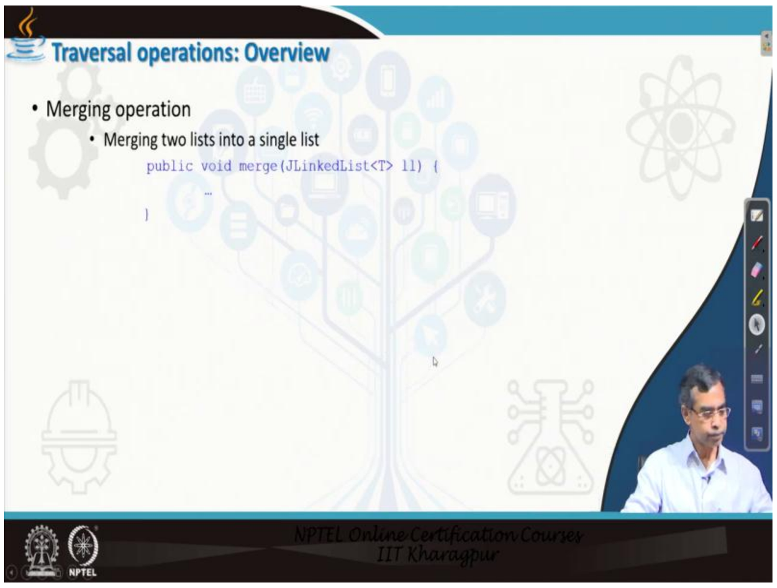
- Methods
  - Insertion operations
    - Insertion at front `public void insertFront(<T> data) { … }`
    - Insertion at end `public void insertEnd(<T> data) { … }`
    - Insertion at any position `public void insertKey(<T> key, <T> data) { … }`
  - Traversal
    - Printing the collection `public void printList() { … }`
    - Reversing the ordering of elements `public void reverse() { … }`
  - Merging operation
    - Merging two list into a single list `public void merge(JLinkedList<T> ll) { … }`
  - Deletion operation
    - Deletion from front `public T deleteFront(<T> data) { … }`
    - Deletion from end `public T deleteEnd(<T> data) { … }`
    - Deletion from any position `public void insertKey(<T> key, <T> data) { … }`

NPTEL Online Certification Courses
IIT Kharagpur

Now what is idea about merge method that we want to discuss about? You see this is the, this is the name of the method is merge, it does not return anything, void. And here is basically the JLinkedList T ll so basically we have to pass as an argument one list and this merge method can be called for a list. This means say list 1 and this is list 2 so it should call the method merge for the list 1 giving the argument list 2. This means that list 1, list 2 will be appended after list 1.

On contrary if you can write, have the two list, list 2 and list 1, and if you call the merge method for list 2 passing the argument as list 1 then here list 1 will be appended after the list 2. So it is basically whether l1 will be added after l2 or l2 will be added after l1, either way you can call it. Now here actually the idea is that how the method can be, can be declared and can be defined. So we have idea about how the method should be declared, what is the signature of the method? Now let us proceed further to understand the actual code how it look like.

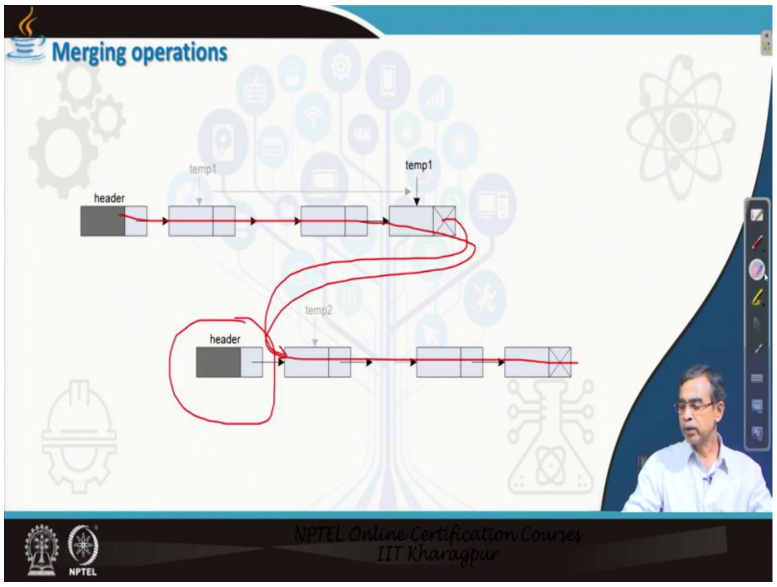
(Refer Slide Time: 48:38)



Traversal operations: Overview

- Merging operation
  - Merging two lists into a single list

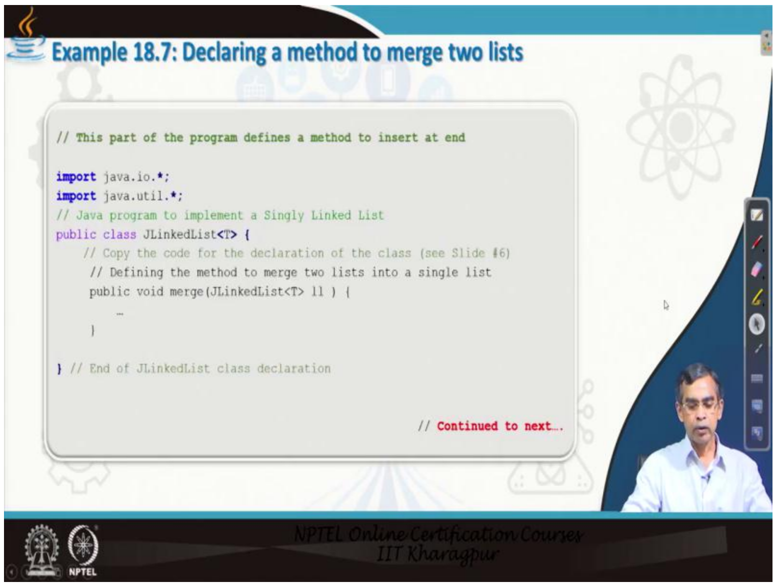
```
public void merge(JLinkedList<T> ll) {
        …
}
```


Merging operations

So this is the signature that means this is the name of the method and then body that we are going write and the linked list as you know, very simple what is called the manipulation of the list. So we have to come to the first node of the temp. So that temp2 should be filled with this one. So that means temp1 dot next is equals to temp 2. That means this basically points.

So this way this node will be here, merge. But this header node is a problem. So what we should do that? We should make free this header node. Now to get a new node there is a new command is there, new operator is there in Java to make a node free, that means we want to return to the pool of memory manager or bank then just free the node. So free header this means it will return to the memory manager, memory bank. So this is basically the idea about, now let us see the code which actually how it look like this.

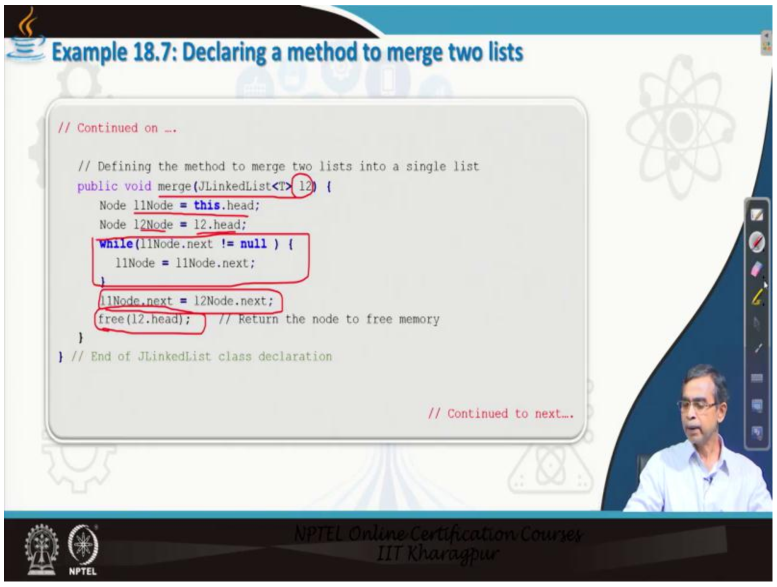
(Refer Slide Time: 49:32)


Example 18.7: Declaring a method to merge two lists

```
// This part of the program defines a method to insert at end

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 Slide #6)
     // Defining the method to merge two lists into a single list
     public void merge(JLinkedList<T> ll ) {
         ...
     }

} // End of JLinkedList class declaration

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

The code is, okay we want to add one more method. This method, we want one more method into the, our linked list declaration that is going on. So this method again we want to add, the merge method.

(Refer Slide Time: 49:47)



Example 18.7: Declaring a method to merge two lists

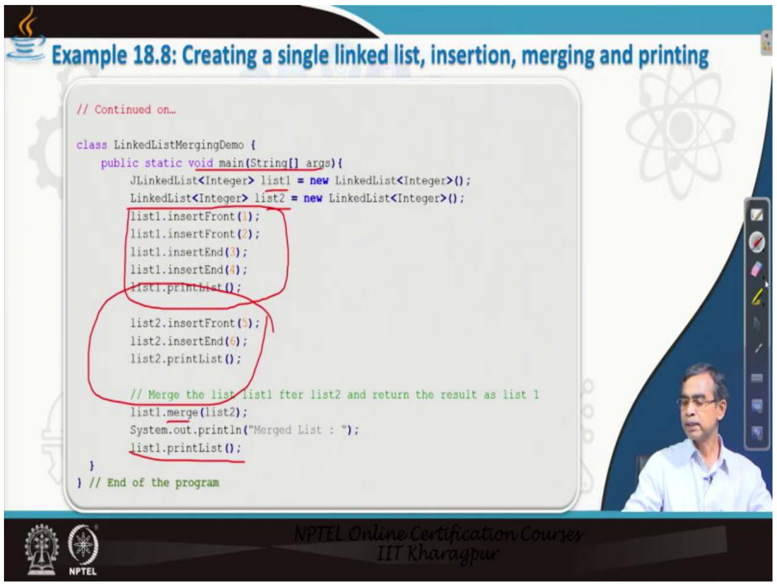
```
// Continued on ….

   // Defining the method to merge two lists into a single list
   public void merge(JLinkedList<T> l2) {
      Node l1Node = this.head;
      Node l2Node = l2.head;
      while(l1Node.next != null ) {
        l1Node = l1Node.next;
      }
      l1Node.next = l2Node.next;
      free(l2.head);     // Return the node to free memory
   }
} // End of JLinkedList class declaration

                                          // Continued to next….
```

Now code, actually a code, this basically managing the links that we have always mentioned. So this is a method l2, we want to append l2 after the call method. Now here suppose node l1 is the current node so this dot head is basically current node. And node l2, l2 head, this basically is the first node of the node l2 which has been passed here. So you can have the two pointers, one pointer is basically the header of the first node and then second is the, we see first node of the next, second list.

Now here you see first we have to go to the last node of the list l1. So this is a simple traversal because we want to add the list at the end of the last node of the first. So that is why we have to do it. And then this is the simple link management as I told you, l1 node dot next, this is the last node, as l2 dot next this is the first node of the second list.

And then we have to make the header free. So this is the method free so that we can return the header which is no more required for the second list, return to the memory bank. So this is a simple idea and you can note it how interestingly you can write the code and it can work. Now again you can write the master program and call all these methods there

(Refer Slide Time: 51:20)



Example 18.8: Creating a single linked list, insertion, merging and printing

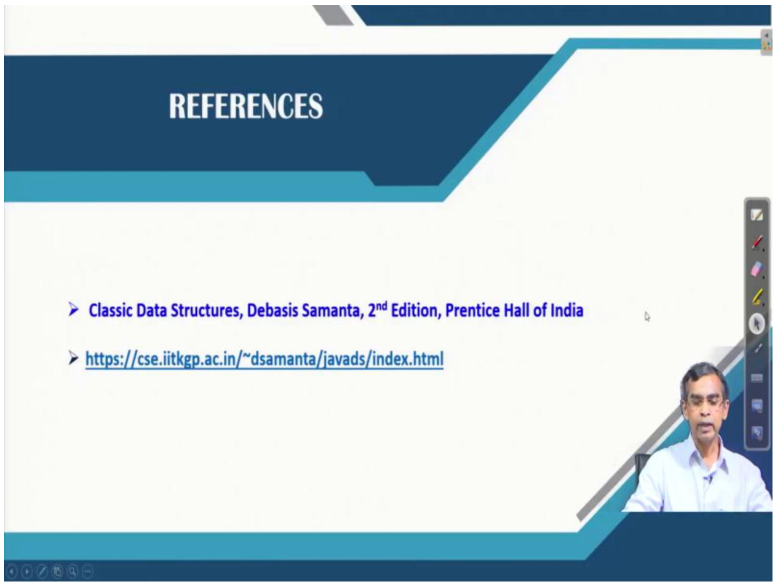
```
// Continued on…

class LinkedListMergingDemo {
    public static void main(String[] args){
        JLinkedList<Integer> list1 = new LinkedList<Integer>();
        LinkedList<Integer> list2 = new LinkedList<Integer>();
        list1.insertFront(1);
        list1.insertFront(2);
        list1.insertEnd(3);
        list1.insertEnd(4);
        list1.printList();

        list2.insertFront(5);
        list2.insertEnd(6);
        list2.printList();

        // Merge the list list1 fter list2 and return the result as list 1
        list1.merge(list2);
        System.out.println("Merged List : ");
        list1.printList();
  }
} // End of the program
```

And then the code will look like this. So this is another master program which basically exhibit the operations that we have so far added into the linked list. Now in this, in these methods we see the main, we are discussing. Two lists we have creating, list1 and list2, all are of type Integer that you should do. You cannot do in other way, one is integer, one is string. Then it will give a compilation error.

Then the different way that we can add so this basically creating the first list l1 and then we also creating the second list adding some element into it. So two list is ready. Then we call the merge method for the list1 passing argument list2 that mean list2 will be added at the end of list1, and then printing the list l1. It will basically print after adding.

Alternatively, you can call again list2 dot merge list1, passing this one, so is the opposite way it can. So this basically the master program if you run you will be able to see how the list can be created, two lists can be created, how one list can be added after the another list and vice versa.

(Refer Slide Time: 52:32)



REFERENCES

- Classic Data Structures, Debasis Samanta, 2nd Edition, Prentice Hall of India
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html

So this is the code that we have think about, insertion operations. Now other operations namely other traversal operations like reversing, and deletion of a node and other operations we will discuss in our next class. I hope you have understood it but my advice is that you can write the code, practice it and for code and everything you can follow this link where you can find all codes that we have used here and then slides and everything you can have this link so that you can enjoy your reading and learning also.

So this is about the first, what is called the phase of programming, part, first part of the programming, part 1 programming concerning linked list manipulation. In the next we will discuss about other operations related to this. This is the second part of the programming. Thank you very much. Thank you for your attention.