

**Data Structures and Algorithms using JAVA**  
**Professor Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 25**  
**Programming for Queue**

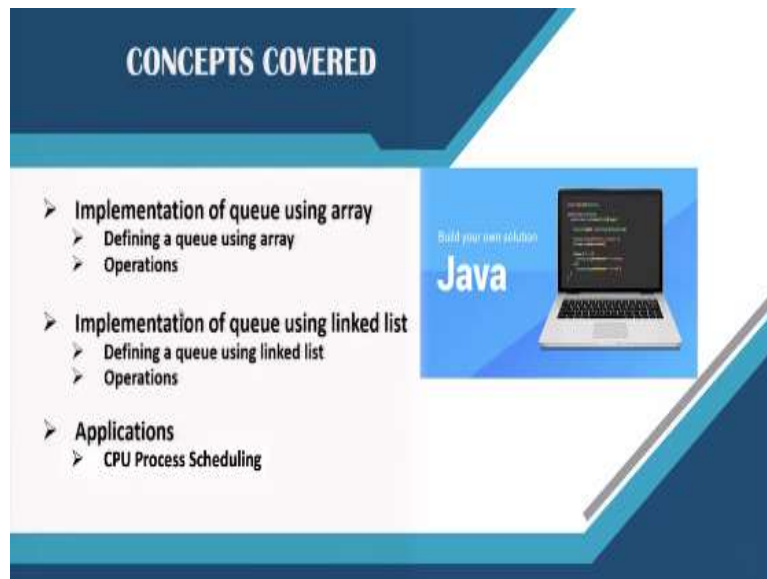
(Refer Slide Time: 00:32)



Today, we shall discuss about Queue Data Structures mainly we will focus how a queue structure can be programmed. Now programming is an important what is called the activity for any data structure actually this is because to earn the skill in any programming we have to exercise two things. One is algorithm and data structure. So, logic behind any programming better can be exercised if we try to implement an algorithm using a data structure.

So this is why we give emphasize on programming. Queue is also now a exception. In the last video lectures, we have learned about queue data structures in details. So this is basically theoretical point of view. Now today there is a practical point of view and this discussion we will continue in the next lectures as well as but in the next lecture we will try to understand the JAVA utility support in the form of JAVA collection framework so that queue can be utilized in your program.

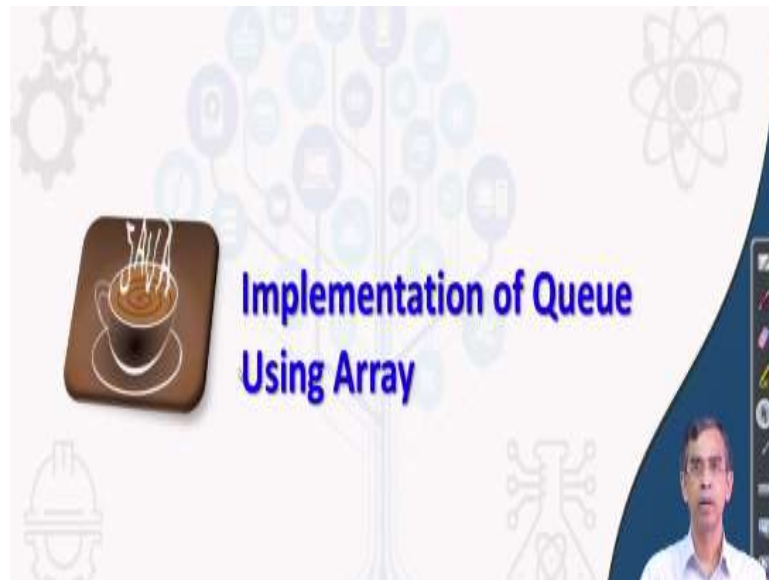
(Refer Slide Time: 02:08)



So, let us start today's topic. Today mainly we will discuss about how a queue can be maintained. So far the maintenance of queue data structure is concerned there are two ways usually a programmer can think for one is using an array. So, using an array how a queue structure can be represented. Then we have also learn about linked list. So here is an application of the linked list so that a queue data structure can be implemented.

So two representations namely array based and linked list based for the queue data structure and learning a data structure can be complete if we see how actually data structure is applicable to solve some problem. So, today we will discuss using linked list representation of queue data structure for an application called CPU process scheduling which is basically very important application usually followed in any operating system implementation.

(Refer Slide Time: 03:26)



So now let us come to the discussion of implementation of queue data structure using array. Now like any data structure programming here also we will exercise a generic implementation of queue. Now let us see how a queue data structure can be implemented using the generic concept.

(Refer Slide Time: 03:50)

The slide displays the following C++ code for a queue implemented using an array:

```
// This program shows how a queue can be defined using an array
class QueueA {
public:
    QueueA(): data{
        int front, rear;
        int length;
        QueueA(T* a) {
            data = a;
            front = -1;
            rear = -1;
            length = a.length;
        }
};
```

Handwritten annotations on the slide include:

- A red circle around the class name `QueueA`.
- A red box around the initialization of `front = -1` and `rear = -1`.
- A diagram of an array with indices `0` to `n-1`. A red arrow labeled `R` points to the `n-1` index, and another red arrow labeled `F` points to the `0` index.

So, first of all we have to define a generic class. Now in this program as you can see we define a generic class. The name of the class as you have given here queue A, A stands for array representation so it is basically queue using array and here you see it has a generic type T this means that queue that we are going to define can hold any type of data integer, it can be character, string, even the user define data type like this.

Anyway essentially a queue is basically a storage space, so we have to define an array to store the elements in the queue. So here we will define the array, the name of the array where the elements will be stored is data and it is basically of type T, T can be anything as I have already told. Now a queue is characterized with two pointers the front and rear, so we define the two pointers as integer because they are basically indexing and here I want to mention one thing.

So this is basically is an array of finite size and in this implementation we will assume that the index starts from 0 to say some numbers n minus 1. So this is basically index so this means that the front and rear will move from 0 to n minus 1. However, in order to mention that queue is empty at any instant what actually we will see is that the front equals to 0 and rear equals to minus 1.

Now here I am telling you why the two pointers front and rear I can say like this as you know queue is basically is a first in first out data structures. So there are two pointers rear so let it be R, R stands for rear and another pointer let it be front F. So here basically insertion and

deletion should be at two different ends. So, the element next where it needs to be inserted it basically at R.

So, if the next element comes the next element will be added where exactly R points to the next one. So this basically is the next location for the next element to be added into the queue so next to R. On the other hand if we want to delete an element from the queue it should be deleted exactly from the location F, F is the front pointer. So before insertion an element so R needs to be implemented by 1.

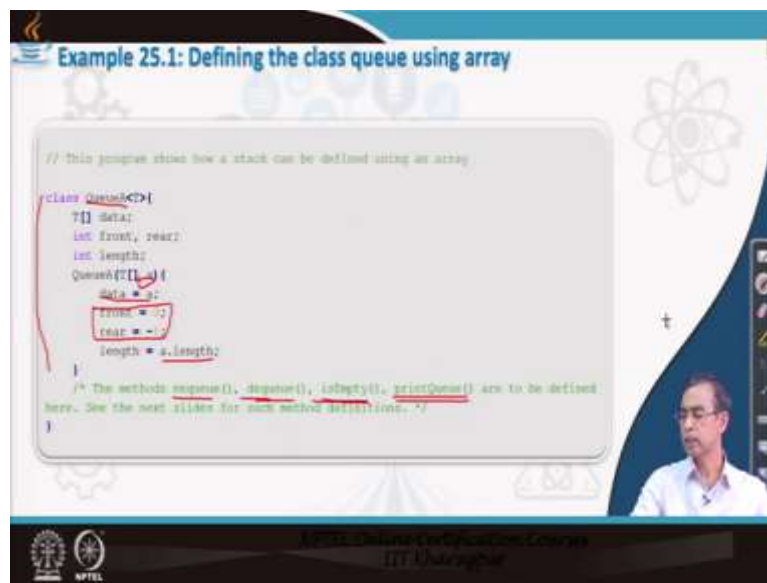
And again after the deletion of an element from the queue the F will be incremented by 1 and you can see both R pointers and F pointers move from this direction to this direction. So a queue will be treated as empty queue if we say front equals to 0 that means it is here that means queue is ready to remove an element from here and rear equals to minus 1 so rear is basically here.

So this is basically a check that we can think for understand whether a queue is empty or not. Now we can say a queue is full if we say R reaches to here if R reaches to here and next time if new elements come for inserting into it then we can say that queue is full overflow. Now again as the front and rear moving this direction so sometimes R is here, queue is full and front remove all elements.

And come here then at that point what you can say after removing this front, front will be increased by 1 so that means front equal to N. So, if we say that front equals to N then again we can reset this front and rear as 0 and minus 1 respectively to indicate that queue is now empty and again we can repeat the procedure. So this is the logic that we should follow and accordingly we can implement the different methods.

The methods namely en-queue and de-queue in order to maintain, in order to perform the insertion and deletion of elements from the queue.

(Refer Slide Time: 09:23)



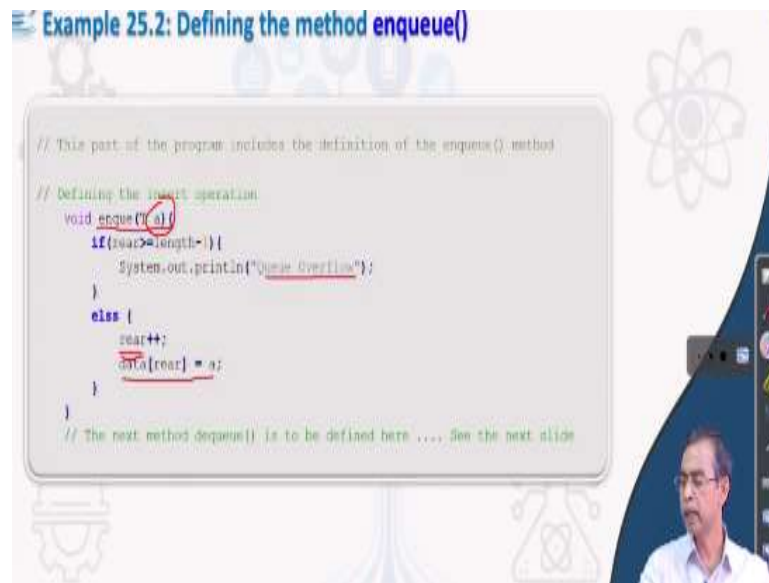
So with this consideration we initially define a queue passing an array assuming that A is an array passed so we can load this queue with an array of elements, this a array of elements maybe null also in that case this data will be null nothing.

Whatever be if you can if you do it accordingly we can write that front and rear is basically 0 and minus 1 respectively to indicate that at the moment queue is initially empty and a dot length if a is empty so it is basically null. So this is the constructor that is there in the generic class declaration and we have to add methods in order to make the class declaration complete. So the methods those are required is en-queue, de-queue is empty and print queue.

En-queue is for inserting an element into a queue, de-queue is for deleting an element from queue is empty to check whether queue is empty or not and print queue is basically to print the current elements those are there in the queue. So, these are the 4 methods that we are to add one by one.

So, let us start first en-queue operation we will implement each operation one at a time and write a master program to check whether the operation is working or not. And then we will add our next method and so on. So let us start defining the en-queue operation first.

(Refer Slide Time: 11:08)



```
// This part of the program includes the definition of the enqueue() method

// Defining the insert operation
void enqueue(int a)
{
    if(rear >= length-1)
        System.out.println("Queue Overflow");
    else {
        rear++;
        data[rear] = a;
    }
}

// The next method dequeue() is to be defined here .... See the next slide
```

Now here enqueue operation is very simple here I have defined one operation for you, you can check these operations, you can copy this code run in your machine and you can try to see how it is working I will come to how the operation can be tested by writing a master program here. So this is basically enqueue as I told you insertion and so far the insertion is concerned it should be added some element at the rear actually.

So that is why here you see if rear is greater than length minus 1, so that means if rear goes to at the end then we can say that queue is overflow no more insertion is possible. So enqueue will fail. If it is not like that then what we should do that we should increment the rear pointer that means go to the next one and then we can add the elements which needs to be added there a supposed to be added into the queue there.

So this way it will complete the insertion operation and you can check all these methods is valid even if the queue is empty and queue is full that is all we have checked and if queue is not full this will do it. So in all situations this method if you can call for a queue it will work I hope, so you can verify it. So this is the enqueue operation.

(Refer Slide Time: 12:43)

```
// This part of the program includes the definition of the dequeue() method

// Defining the delete operation
T dequeue() {
    T x = null;
    if (isEmpty()) {
        System.out.println("Queue Underflow");
        return null;
    }
    else {
        x = data[front];
        front++;
        return x;
    }
}

// The next method isEmpty() is to be defined here .... See the next slide
```

Now let us come to the next the dequeue operation. Here is the dequeue operation that we have defined. Dequeue operation does not require any argument to be passed unlike enqueue operation however dequeue should return a value is basic value of the type that basically queue stores so T. Now here you see so the first check that okay fine, T we will define x is a temporary actually if it deletes an element then it should return that element.

So x is basically the elements to be return so initially x is null means it does not return anything actually in that sense so null. Now before going to check the deletion operation dequeue we have to check first that it if is empty because if a queue is empty then we do not have to perform any operation, dequeue operation. Dequeue operation is not valid for any empty queue so we just simply return null.

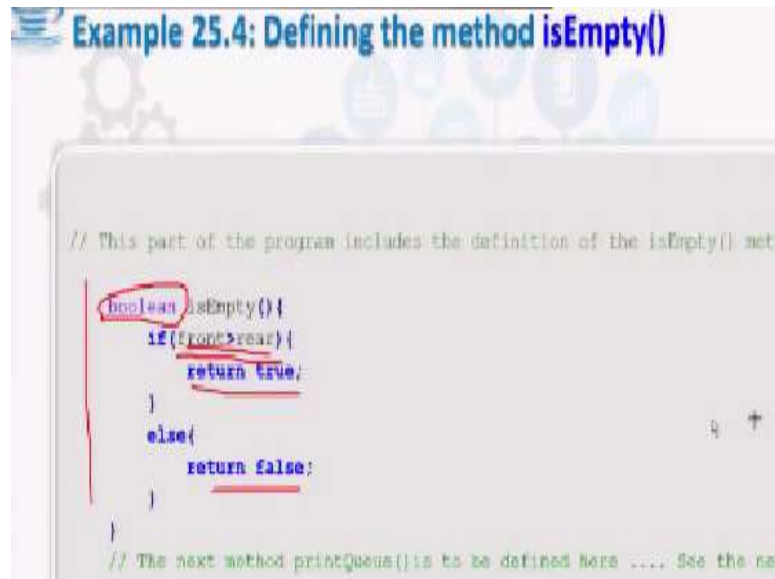
If it is not empty then we have to perform these operations. Now you see what we are doing here as I told you, so deletion will take space from the front one and this way first in first out will be implemented. So here at the front most element will be copied into x this means x needs to be returned and then front will be implemented to the next location and we return x, so this is a pretty simple the operation that we can think for the dequeue.

So this operation is basically the dequeue operation using the array representation and as we have already told you if even it is the queue is full that you can check that this method is also still valid. So you have to test the operation under 3 different situation, empty it is just (()) (14:47) empty and it is full and you can check it. Anyway so we will definitely check it now here you can see we have used one isEmpty method.



We have not declared is empty method earlier, but anyway we will go for declaring is empty method.

(Refer Slide Time: 15:08)



```
// This part of the program includes the definition of the isEmpty() method
bool isEmpty() {
    if (front > rear) {
        return true;
    }
    else {
        return false;
    }
}
// The next method printQueue() is to be defined here .... See the next slide
```

Now let us declare is empty method and it is very simple again. So is empty method will return true or false so that is why return type is Boolean. Now here we are checking whether a queue is empty or not and this is the condition that basically stands for indicating queue is empty the condition is that if front greater than rear, so whenever the front exceeds the rear pointers then it basically check that it means that queue is empty.

So it is return true otherwise it return false. So this is a pretty simple method and the condition needs to be followed like this one and that is all so this basically implement 3 methods and then another method that is remaining this is basically the print queue. Print queue is pretty simple at any instant whatever the elements in between front and rear we need to print.

(Refer Slide Time: 16:03)

```
// This part of the program includes the definition of the printQueue() method
// Defining an operation to print an entire queue content
void printQueue() {
    if(!isEmpty()) {
        for(int i = front; i <= rear; i++) {
            System.out.print(data[i] + " ");
        }
        System.out.println();
    }
}
// End of the definition of the class Queue
```

So that method is again very pretty simple it is here. The print queue method we have defined it does not require to return anything so return type is void and it does not require any argument to work with so it is null, void and now if not empty definitely there is no point of printing element for an empty queue so that is why we have to perform a check if it is not empty then only for int i equals to front and i less than equals to rear.

So basically all elements in between front and rear including both are basically the elements in the queue at any moment so we just simply go on printing those elements from the queue and finally we come to end. So this is basically the print queue method pretty simple and this completes all 4 methods declaration for the array implementation, implementation of queue using array.

Now we are in a position to test the data structure that we have implemented the generic data structure of queue implemented using R. So for this purpose we have to write a master program. The master program should include enqueue, dequeue is empty, print stack whatever be the order you wish and you can check. So this way you will be able to test for an empty queue, for non-empty queue and full queue and performing the operations here.

(Refer Slide Time: 17:40)

```
/* This is the main program, illustrating the usage of the class defined. The
 * main include the package, where this program is defined. */

class QueueImplementationDemo {
public static void main(String[] args) {
    Integer arr[] = new Integer[2];
    Queue<Integer> q = new Queue<Integer>(arr);

    q.enqueue();
    q.printQueue();
    q.enqueue();
    q.printQueue();
    q.enqueue();
    q.printQueue();
    q.dequeue();
    q.printQueue();
    q.dequeue();
    q.printQueue();
    q.dequeue();
    q.printQueue();
}
} // End of the demo class
```

```
/* This is the main program, illustrating the usage of the class defined. The
 * main include the package, where this program is defined. */

class QueueImplementationDemo {
public static void main(String[] args) {
    Integer arr[] = new Integer[2];
    Queue<Integer> q = new Queue<Integer>(arr);

    q.enqueue();
    q.printQueue();
    q.enqueue();
    q.printQueue();
    q.enqueue();
    q.printQueue();
    q.dequeue();
    q.printQueue();
    q.dequeue();
    q.printQueue();
    q.dequeue();
    q.printQueue();
}
} // End of the demo class
```

Now here is a master program that I have given an idea for you, but you can use the program, but you can add any order regarding in any order you can perform insertion, deletion into the queue. Now you see this program, this program is basically queue A implementation demo and here we declare array. Array is basically array of integers type it is declared initially the size is 2 we have intentionally made the size 2 you can made it little bit larger size anyway so no issue it is there.

Then what we do is here this is the queue A our array implementation for the queue and here we are passing the generic type that integer that means we want to store integer value into the queue and this is here you see we initialize the queue passing array. Array is basically our

elements so initially this array is null because it is just allocated so there is no element at the moment.

So it is basically empty array we can say because this array is declared, but not initialized by any elements so it is like this. So this basically you can check is basically call the constructors as we have defined there in the queue implementation in our class declaration and basically call the constructor enqueue this is basically our queue actually object, this is a queue actually right, and this queue is ready to allow a programmer to insert and delete and print is empty test everything.

Now here we are performing certain operation as we see `q dot enqueue 1` so basically enqueue method we are calling here passing an argument one that means we are inserting the first element into the queue which is 1 and after inserting we are just simply print queue. Next, we are inserting another element and print queue and so on and so on. Now, here up to this it is okay no problem, but you note that our size of the array is 2.

So whenever we come here then we can see the queue is full because it is already full. So print queue will print only 1 and 2, but it will not insert any element into the queue. Now let us dequeue so of this dequeue is valid again this dequeue is also valid again if you come to this dequeue then here again you will see this is not valid this is because in this case queue is already empty.

So here the two cases are not valid because this is full and this is empty and all other cases this is basically not empty situation. So this way we have written the master programs so that you can test all situations. Now you can again re-run the program by changing these two as a little bit larger maybe 5, 6, 10 and then performing several operation if possible you can write a for loop and do many insertion then many deletion and so on and you can check it.

So programming that you can continue with this what is called the master program and the declaration that we have defined for the queue using array to test your program is queue. So this is must actually otherwise you will not be able to cope with the programming and particularly data structures and algorithm in order to implement your program so this is important.

So this is the case that we have discussed about queue implementation using array now we will repeat the same thing, but using another data structure namely linked list. Now as you

see using array if we implement a queue then it is limited to the size of the array. So this is one basically is a disadvantage for this method otherwise it is very fast actually. So, if you implement a queue using array it is really very fast, but only limited that its size is limited static or bounded like.

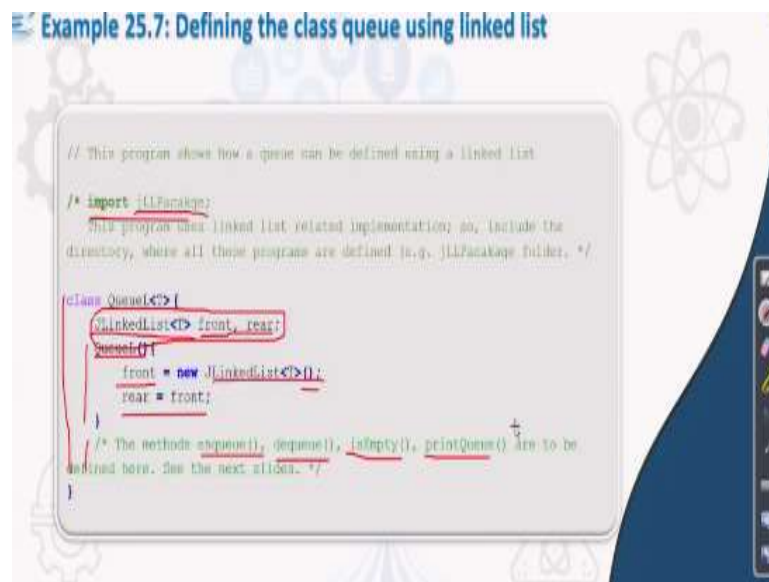
On the other hand if you use linked list because linked list can allow you to add as many as elements you wish that means that it is unbounded or actually it is infinite capacity or it can grow dynamically that is why the linked list is more preferable and you will see the JAVA collection framework basically follow linked list implementation for their queue not that array, but array queue has many other significance particularly speed point of view.

(Refer Slide Time: 22:48)



Our next task is basically implementation of queue using linked list. Now like the array implementation here also we should declare a queue structure using linked list and that is also generic. So, a generic what is called the class needs to be declared first with the fields and the constructor this is the usual procedure and then different method related to enqueue, dequeue and everything.

(Refer Slide Time: 23:19)



```
// This program shows how a queue can be defined using a linked list
/* import JLinkedList;
This program uses linked list related implementation; so, include the
directory, where all these programs are defined (e.g. JLinkedList folder. */

class Queue<T> {
    JLinkedList<T> front, rear;
    Queue() {
        front = new JLinkedList<T>();
        rear = front;
    }
    /* The methods enqueue(), dequeue(), isEmpty(), printQueue() are to be
    defined here. See the next slide. */
}
```

Now let us proceed. Now so here we can tell one thing linked list need to be used here so we should have the linked list implementation first. Now already we have discussed about how the linked list can be defined for, okay the program is with us so that programming that all the class declaration for the linked list should be imported in this program, so here assuming that this class is defined in this package so we should import this package first.

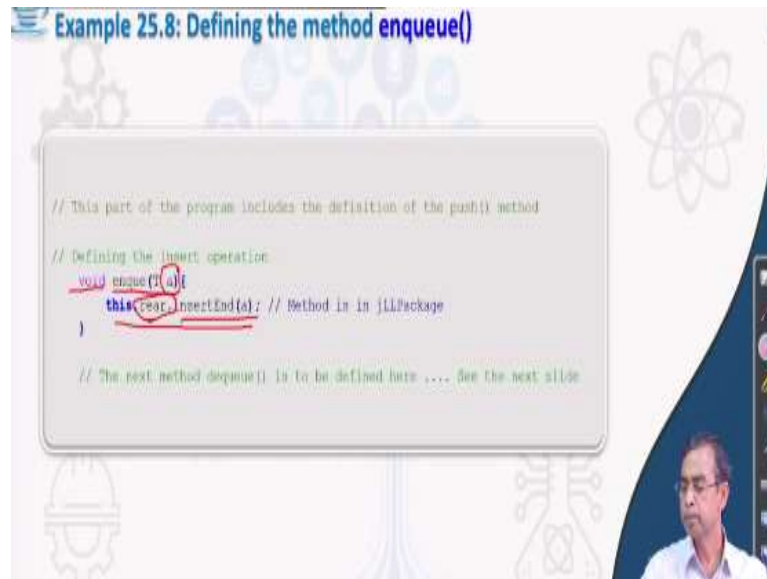
So this means that all the previous programming for linked list is now added into this program and as you know in this package the class J linked list is defined that is also defined generically. So what we are doing is basically we are creating two pointer, the front and rear of type J-linked list. Now these basically pointer to indicate that they will point to two nodes the front and rear.

Now in case of queue as I already you and we have already exercised for the array is there are two pointers front and rear the rear is a position where the deletion will take place and front is the position where insertion will take place. So, rear is basically at the end of the queue or end of the queue we can say so this means that the insertion will takes place at the end and front is basically front or just next to the head. So deletion will takes place from front.

Anyway, so we just declare a constructor for this class queue L this is basically the constructor to initialize a linked list that basically actually a queue for you. So here we just initialize linked list constructor initially it is empty and front is basically is the front most location it is basically you can say header. So this basically creates a front and rear is equal to front initially rear and front are the same it is a header.

Header indicates that initially queue is empty with the linked list implementation. Now so we are now ready the queue is declared our next task is basically to add method into this class declaration. So in this position we will include the declaration or definition of the method the enqueue, dequeue is empty, print queue one by one. So, if we do it our task will be complete. So let us see how we can add the method one by one.

(Refer Slide Time: 26:04)



```
// This part of the program includes the definition of the push() method
// Defining the insert operation
void enqueue(T a){
    this.rear.insertEnd(a); // Method in java package
}

// The next method dequeue() is to be defined here .... See the next slide
```

So this is basically the enqueue method it is very simple enqueue and it does not return anything so wait and we have to pass the element to be added here and this is very simple as you see enqueue operation is nothing insert at end. So insert end it is basically defined for the linked list implementation that is all. So it is basically we just insert an element at the rear actually that is fine. So this is basically we will complete the insertion.

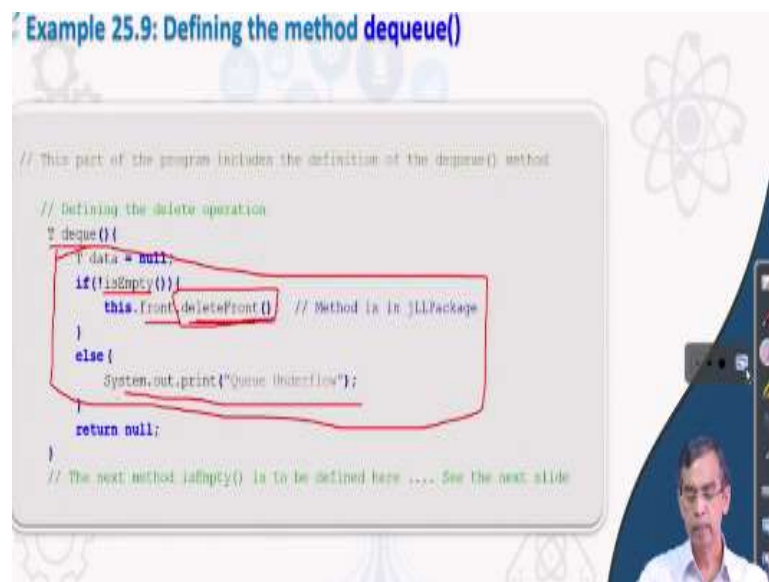
(Refer Slide Time: 26:40)

**Example 25.9: Defining the method dequeue()**

```
// This part of the program includes the definition of the dequeue() method

// Defining the delete operation
T dequeue() {
    T data = null;
    if (!isEmpty()) {
        this.front.deleteFront(); // Method is in JLLPackage
    }
    else {
        System.out.println("Queue Underflow");
    }
    return null;
}

// The next method isEmpty() is to be defined here .... See the next slide
```



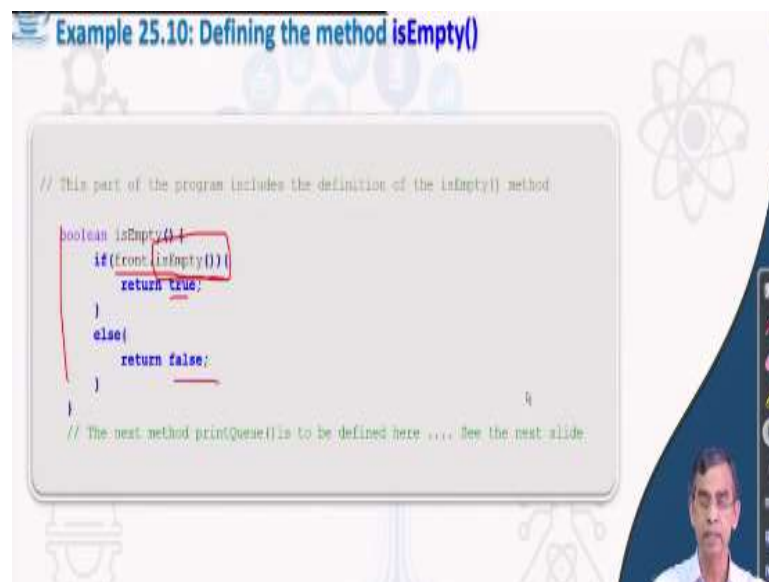
Likewise the dequeue operation. The dequeue operation is basically here you note one thing need not for the enqueue operation we do not require to check whether queue is full or not because queue will not full so far the linked list implementation is concerned, but for the dequeue operation we have to check whether the queue is empty or not. Queue is empty means when you can say that list, linked list is empty actually.

So there is a test is required, so this test is here to check that if empty or not again in empty method we can define for the linked list is very easy and if it is not empty and then if it is empty then definitely we will print that this linked list is empty or underflow. If it is not empty then we see delete front that means this method is defined for the linked list class. We can call the delete front operation at the front pointer that means in the front so it will delete this element note and that is all.

So it deletes and definitely return here we can write return the element the delete front itself return the value so this way it will return automatically. So this basically completes the dequeue operation for you and, so this is a dequeue operation.



(Refer Slide Time: 27:59)



```
// This part of the program includes the definition of the isEmpty() method

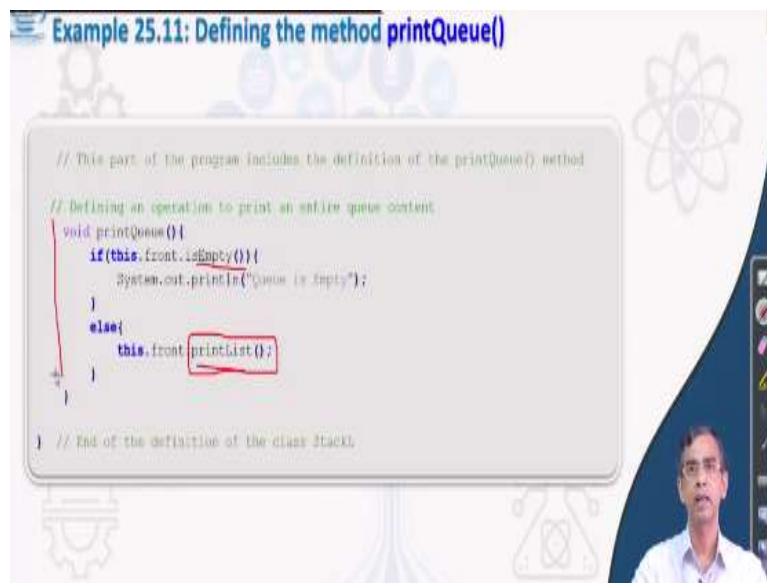
boolean isEmpty() {
    if (front == null)
        return true;
    else
        return false;
}

// The next method printQueue() is to be defined here ... See the next slide.
```

Now let us come to is empty operation is empty operation is pretty simple it basically is a front is empty. This method is again defined for the linked list class that we are using. So simply call this method to check if it is empty then true else it is false. So this method is very simple here and you can see the method is simple because we have already linked list implementation and we are just using this linked list implementation for our queue.

And that is also one application of linked list actually. We have also experienced application of linked list to implement our stack also. So, this basically complete and finally the print.

(Refer Slide Time: 28:35)

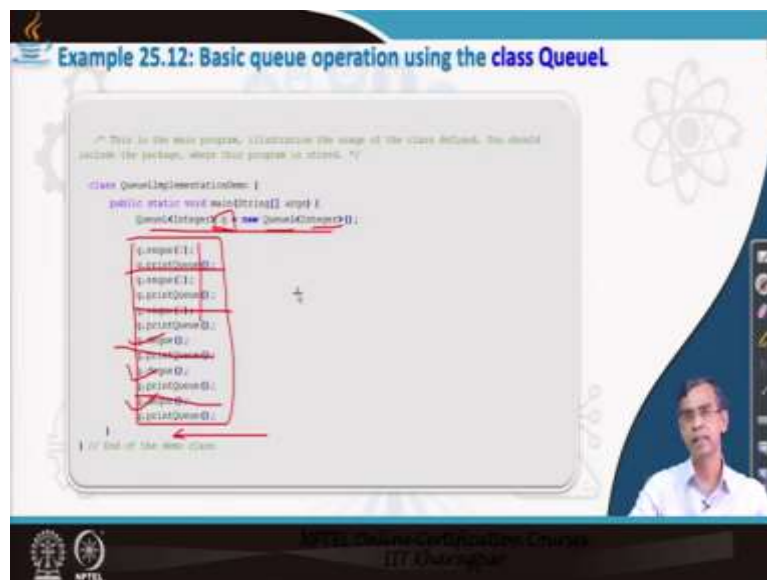


```
// This part of the program includes the definition of the printQueue() method
// Defining an operation to print an entire queue content
void printQueue(){
    if(this.front.isempty()){
        System.out.println("Queue is empty");
    }
    else{
        this.front.printList();
    }
}
// End of the definition of the class StackL
```

Print method is same as the printing the linked list and obviously the print method we have to check if it is empty. If empty nothing to print, if it is not empty print list method, this method again define for the linked list class. So call the method and it is done for you. So these are the programming what is called the looks that you should have in order to implement the queue structures and accordingly it implies.

So we have done all 4 methods and added one by one into the queue L that is the generic class implementation for the queue structure in this case using linked list and now we are ready to have a master program.

(Refer Slide Time: 29:15)

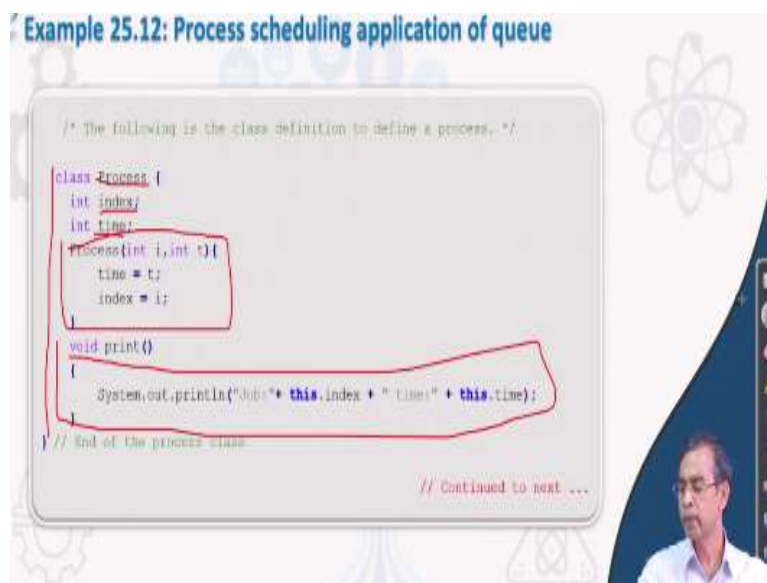


So here is a master program and here we can see here we are using, we are declaring a queue that is basically it is our queue using queue L, queue L basically our class declaration and this is an integer that means we want to store integer into this one. And this completes your declaration about a queue.

A queue is ready and we can perform like earlier the different insertion and deletion in any order. So we have find this order and everything and you can check that here actually we have to test whether is empty. So if you queue 1, queue 2 then enqueue so 3 elements are added and then perform dequeue it is okay, this is also okay, this is also okay so it is here.

But later on if you perform any other dequeue operation here in this case for example you can check it and you will see that it can be gives you underflow that means queue is empty error. Anyway so you can check with different situation you can write being to this program and then you can test that your program is working and this basically completes the linked list queue implementation using linked list. So, we have learned about two ways array implementation and linked list.

(Refer Slide Time: 30:28)



And quickly we will cover one application, application of queue. I just want to discuss on application. It is basically process scheduling so here idea is that many process will come and you have to start the process. Now here the policy that needs to be follow is that the first come first serve that means the process which comes first needs to be served first. Now if it is like this then we have to follow a queue structure to maintain this policy. So it is basically call FCFS first come first serve process scheduling.

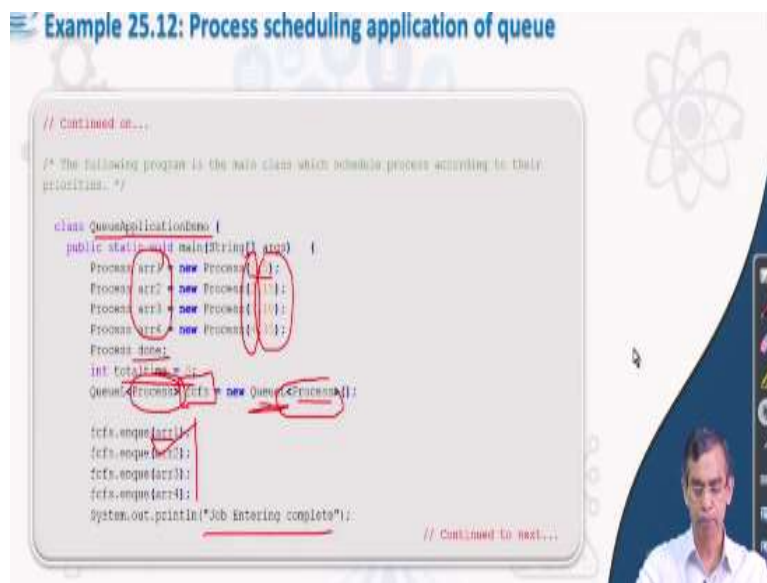
And to implement this scheduling protocol we have to follow one queue. Now again let us consider the linked list implementation of the queue to implement this one. Now before going to use the queue first let us define a process, we can define a process by it is an identity and

what amount of time that it requires. So here with this consideration a process is a class we declare here.

Index is basically idea of the process, time basically how much time it require and this constructor is basically initialize a process and it will basically print what exactly the process it is so it is the process. So this is a pretty simple to define a process. Now we can use this class declaration to define a number of processes, create process instances and then we can add process whenever they are created into array and then implement into the FCFS policy.

So the next part is basically the queue and process will arrive into the queue and then add them into the queue and then serve them.

(Refer Slide Time: 32:12)



```
// Continued on...

/* The following program is the main class which schedule process according to their
priority. */

class QueueApplicationDemo {
    public static void main(String[] args) {
        Process arr[] = new Process[4];
        Process arr1 = new Process(1);
        Process arr2 = new Process(2);
        Process arr3 = new Process(3);
        Process arr4 = new Process(4);
        Process done;
        int totalTime = 0;
        Queue<Process> q = new Queue<Process>();

        q.enqueue(arr1);
        q.enqueue(arr2);
        q.enqueue(arr3);
        q.enqueue(arr4);
        System.out.println("Job entering complete");
    }
}

// Continued to next...
```

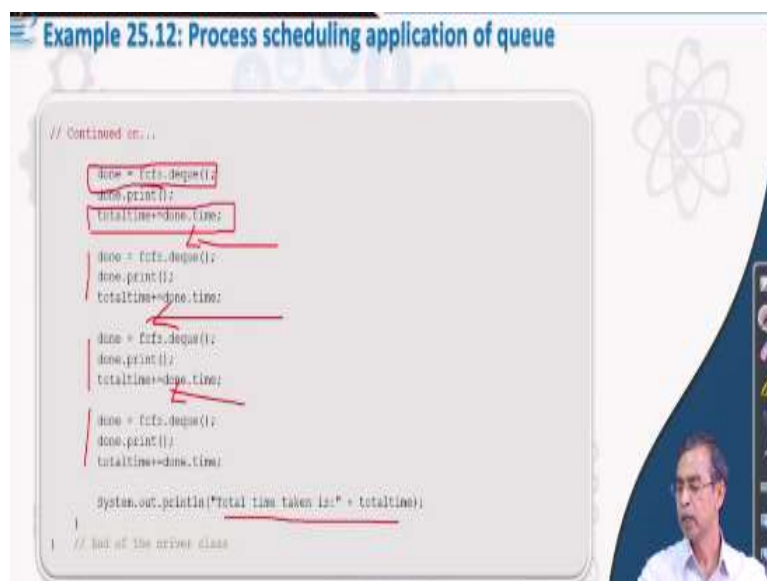
So here is the next part of the program this is basically queue application demo. Here, we can see we can see 4 processes, so it is basically these are the 4 processes and it basically indicates the identity, these are the 4 process identity and this is the time that is required in order to serve the process (()) (32:31) like this one and we use one temporary variable it is called the done.

Totally time actually after this policy that means all process are served then how much time the scheduler takes to calculate initially it is 0. Now here we maintain a queue this queue is FCFS let it be and here we use the queue L that is the linked list implementation of the queue and it is basically process because we want to store process into our queue. So you can see this is a generic thing.

So that process can be added into the queue not like integer or thing that we have discussed earlier. Now all the process have arrived so we can insert or enqueue the process into the queue. So we are entering all the process into the queue, but here we have given an idea but you can see one process can come after this one or you can remove this process or whatever it is there that you can go on.

Initially the process arrives there and the process will be served as a first term they may arrive one will be first and so on and this is basically job comes to the scheduler and now scheduler has to serve the processes. Now here is the next part of the program that we can think for. And this is basically how the process will be served.

(Refer Slide Time: 33:55)



```
// Continued on...
dnode = cfs.dequeue();
dnode.print();
totaltime+=dnode.time;

dnode = cfs.dequeue();
dnode.print();
totaltime+=dnode.time;

dnode = cfs.dequeue();
dnode.print();
totaltime+=dnode.time;

dnode = cfs.dequeue();
dnode.print();
totaltime+=dnode.time;

system.out.println("Total time taken is: " + totaltime);
// End of the driver class
```

As you can see so this basically dequeue basically it is basically serving. Serving is basically by means of dequeue implementation dequeue implement that it is serving done and then total time that is required for the scheduler to serve this process. So this way all processes all processes are served one by one because in the same order it will basically serve. Dequeue will follow in the same order as enqueue takes place.

And then finally it will print the how much time the scheduler takes to serve all the processes. Now this program again you can customize according to the different way in between some enqueue operation can be carried out and then you can check it and you can test it and many more dequeue operation and so on and you can understand and also you can print queue at any instant and whatever is empty you can check, whatever you can do, do.

So here you can see FCFS is the queue which basically helps you to implement your problem or your application or program. So this is the processes this is one application that we have discussed.

(Refer Slide Time: 35:01)



There are many more applications you can find and this is the link you can find many more applications with detail discussion about so that you can test your programming skill to solve each application if time permits. More programming definitely will give more skill and more confidence in your programming endeavor and this is the link that again we have maintained all the concept that we have cover in this video as well all program that we have used here for the demonstration.

So with this things, I would like to stop here for today. Next, we will discuss about implementation of queue using JAVA collection framework that is another interesting topic that is waiting. Thank you for your attention.