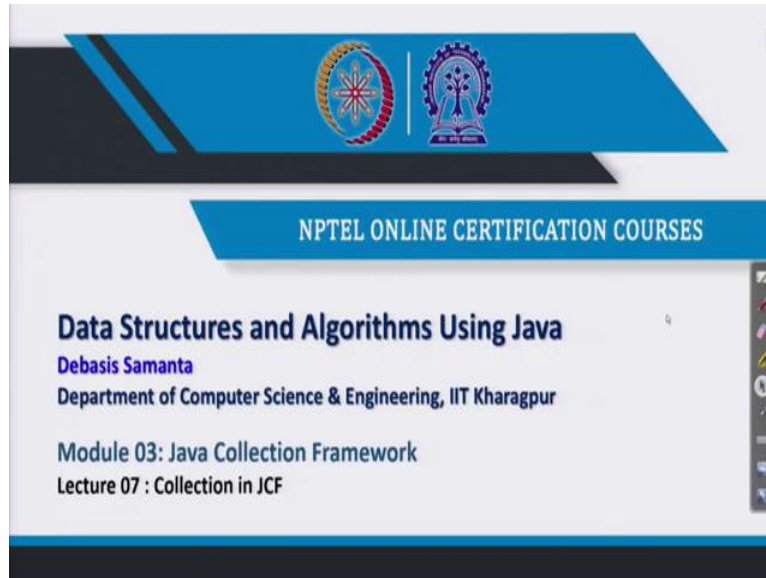


Data Structure and Algorithms using JAVA
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 7 - Collection in JCF

(Refer Slide Time: 00:31)



We are discussing Java collection framework. A brief introduction to the Java collection framework we have already learned in the last video. You have okay, you can recall that there are so many what is called the facilities for the management of different type of data structure. Today, we will take a quick tour to the whole stock of collection framework as it is vast. So we should discuss this topic in three more video lectures.

(Refer Slide Time: 01:19)

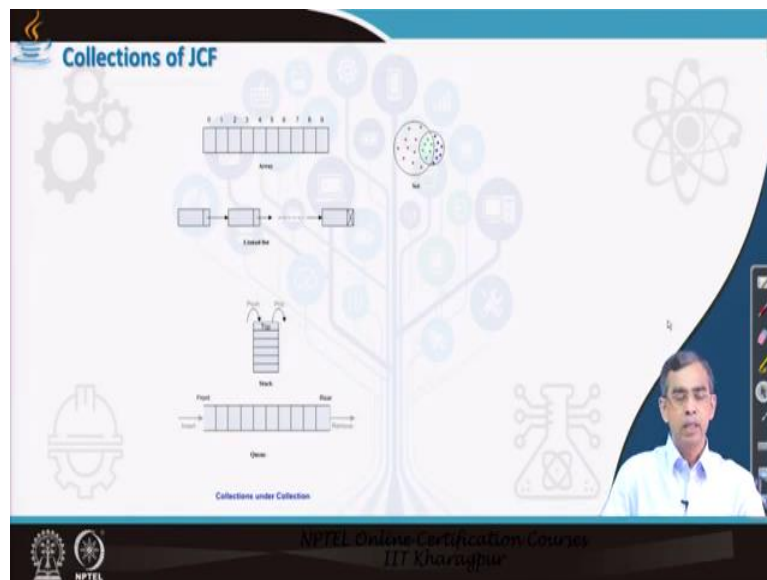


So today, we will start some part. Today, actually, we will discuss about the basic composition of Java collection framework, composition means it includes a large set of interfaces and classes. Today, we will discuss few important interfaces and other interfaces will be discussed in the next video lectures. Other than these interfaces there are certain classes also, those are the main things in the collection framework.

In order to know the classes, basically we are to be familiarized with the constructors and methods in each classes. Now the understanding of all those concept that means what are the different interfaces and classes are very much important for any programmer.

So we have to invest our time to understand these things and as you will see there is no programming at the moment while I discuss these collection framework rather constituents of the collection framework. Programming and everything will be discussed when we will discuss the different data structures in due time.

(Refer Slide Time: 02:44)



Okay, so first let us discuss about the basic composition of collection framework. Now collection actually as we, we are terming the concept collection, actually collection means it is basically is a group of data. Now, so group of data how it can be stored depending on that the different structures are known. For an example, array, so array stores a set of data and in a specific it follows certain principle, so that is why array.

However the same principle is not followed in another collection that is called the link list. So every data structure whether it is array or link list or it is a stack or a queue or in our conventional concept it is set, they have their own policies to store and also their own policy to perform certain operation on the elements those are belong to the collection.

(Refer Slide Time: 04:08)

Collections of JCF

- A **collection** that provides an architecture to store and manipulate the group of objects.
- Java collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- The hierarchy of the classes and interfaces in JCF is quite complex.
- The entire Java Collections Framework (JCF) is built upon a set of standard interfaces, classes and algorithms.

• **Interfaces:**
Set, List, Queue, Deque

• **Classes:**
ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet

NPTEL Online Certification Courses
IIT Kharagpur

So interface and classes those are going to be discussed in this lecture, basically provides the facilities that how a particular type of collection can be maintained, can be managed. Now overall, the collection is as we told you that collection is a very vast thing, this is vast because it covers so many data structures in one bundle.

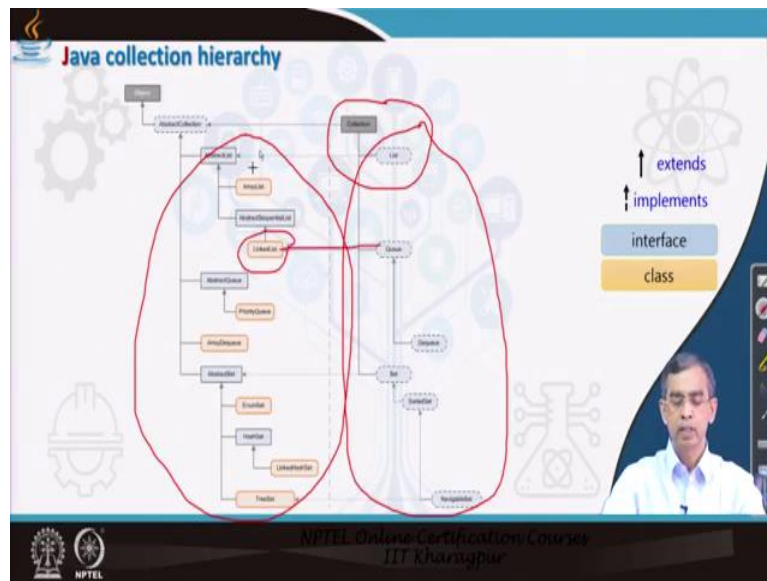
To understand the different facilities so Java provides the concept of interface. As you know interface is basically is a template for certain basically implementation. So interface contains declaration of many fields or methods but body of the methods are not there. So interface gives simple highlights about what are the methods, what these methods return, if you want to call this method what should be the argument and so many things.

In fact, whatever be the interfaces are there they need to be implemented. Implemented means all the methods those are declared there they are to be defined. In Java collection framework all classes basically implements all the interfaces those are there. Now all these classes basically plan to cater to need of a particular data structure, okay.

So actually, all the interfaces again can be categorized to give certain declaration of different functionalities or methods for different data structure like set, list, queue, de-queue, array or lot of varieties. Similarly, the classes which basically implements all those interfaces are there to give the full what is called the life or safe to the facilities.

So they are basically cater to the need of arrays namely, array list class, vectors, link list, priority queue, link hash sets, tree-set and so many things are there.

(Refer Slide Time: 06:43)

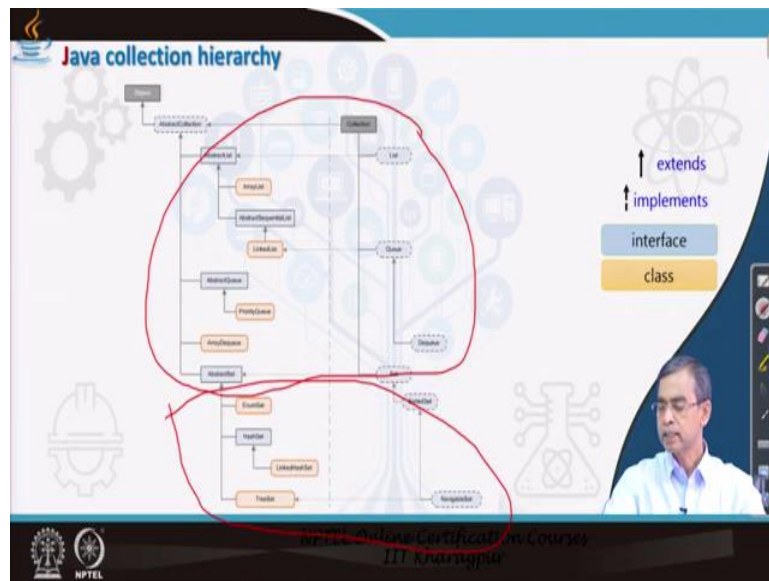


Now overall story really is very complicated and there are certain class hierarchy, the taxonomy of the classes which basically Java collection framework follows. Now as we see this figure, in this figure, we see lot of interfaces which are represented by this kind of relationship among them. Collection is basically is a super-interface basically, so here actually we see collection inherits, list inherits collection, queue inherits list as well as collection, de-queue is basically inherited from the queue interface, like this one.

So interfaces are there follow the same concept as class except that interface contains method without body in them. Now although the interfaces are implemented with separate classes some classes are abstract classes, they are like interfaces but they are abstract. On the other hand, there are certain classes basically implements all this interface.

For example, here link list is a class which implements queue is a interface and so on. So here basically which are the interfaces and which are the different classes is shown in one view actually.

(Refer Slide Time: 08:10)



Now in this lecture, we will discuss all these things and in the next lecture, the next part of the things will be discussed. Now these are the basically called a collection framework. Collection framework consist of this kind of thing and these things are there. So today now let us discuss about first the collection framework which includes the few interfaces in it.

(Refer Slide Time: 08:49)



Interfaces of collections

Interface	Description
Collection	Enables you to work with groups of objects; it is at the top of the collections hierarchy.
List	List extends Collection to handle sequences (lists of objects).
Queue	Queue extends Collection to handle special types of lists in which elements are removed only from the head.
Deque	Deque extends Queue to handle a double-ended queue.
Set	Extends Collection to handle sets, which must contain unique elements.
SortedSet	Extends Set to handle sorted sets.
NavigableSet	NavigableSet extends SortedSet to handle retrieval of elements based on closest-match.

Table 7.1: Interfaces in collections framework

NPTEL Online Certification Courses
IIT Kharagpur

Now let us see what are the interfaces are there in the collection. This table shows altogether all interfaces those are there in Java collection framework. As you see in the list these are the basically interfaces, list, queue, de-queue, sets, sorted set, navigable set and then collection is basically the super, super interface we can say.

Now what these collections are? Again, I repeat all these collections declares certain methods, these methods are basically the functionalities that can be applied to many structure data, data structure that is the concept.

For example, list interface is basically includes all the methods that you can have while you write your program to maintain list data structure, like say link list sort of thing. Similarly, queue, de-queue etcetera. So these are the interface, now I will just okay, quickly summarize different methods those are there because understanding of all those methods really matters for the programmers.

(Refer Slide Time: 10:09)

The slide features a central graphic of a tree with various icons (gears, a coffee cup, a hard hat, a beaker, and a molecular structure) as branches. The title "Interfaces in Collection" is prominently displayed in blue text. A small inset video of a speaker is visible in the bottom right corner. The footer includes the NPTEL logo and the text "NPTEL Online Certification Courses IIT Kharagpur".

The slide is titled "Collection interface" and contains the following text:

- The **Collection** interface is the foundation upon which the collections framework is built because it must be implemented by any class that defines a collection.
- Collection is a **generic interface** that has this declaration:

```
interface Collection<T>
```

Here, T specifies the type of objects that the collection will hold.

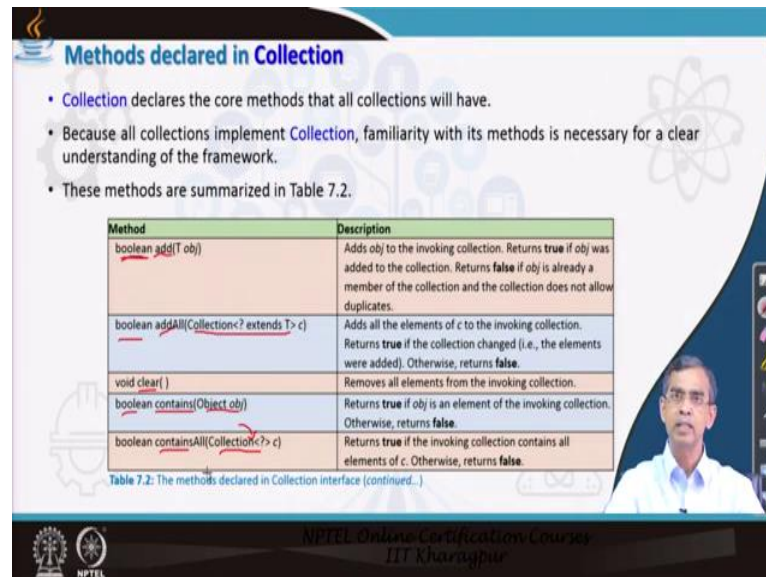
The slide also includes the same tree graphic and speaker inset as the previous slide, along with the NPTEL footer.

So it may be a bit boring to know but if you little bit hold your patience and go through these things, it really give lot of things to your learning process. So I should advise you to take a patience, hold your patience and then check whatever the methods are there although in the this video of a short duration I will try to quickly give you because we have to cover so many things are there.

Now let us first consider the collection interface. Collection is the main or super we can say, it is the top of all interfaces, is basically represents a collection of any type. So that is why this collection is basically is a generic like, so T is basically any type. Any type means if you want to store in your collection integer type of data, so this T will be integer or double or string.

If you want to store the collection of records of type student then T can be student or book or person like this, so this basically this one. So this is the idea about the interface that is a collection and let us see what are the methods are there in this collection interface.

(Refer Slide Time: 11:40)



Methods declared in Collection

- `Collection` declares the core methods that all collections will have.
- Because all collections implement `Collection`, familiarity with its methods is necessary for a clear understanding of the framework.
- These methods are summarized in Table 7.2.

Method	Description
<code>boolean add(T obj)</code>	Adds <code>obj</code> to the invoking collection. Returns <code>true</code> if <code>obj</code> was added to the collection. Returns <code>false</code> if <code>obj</code> is already a member of the collection and the collection does not allow duplicates.
<code>boolean addAll(Collection? extends T> c)</code>	Adds all the elements of <code>c</code> to the invoking collection. Returns <code>true</code> if the collection changed (i.e., the elements were added). Otherwise, returns <code>false</code> .
<code>void clear()</code>	Removes all elements from the invoking collection.
<code>boolean contains(Object obj)</code>	Returns <code>true</code> if <code>obj</code> is an element of the invoking collection. Otherwise, returns <code>false</code> .
<code>boolean containsAll(Collection?> c)</code>	Returns <code>true</code> if the invoking collection contains all elements of <code>c</code> . Otherwise, returns <code>false</code> .

Table 7.2: The methods declared in `Collection` interface (continued...)

NPTEL Online Certification Courses
IIT Kharagpur

There are many methods in fact and again, one thing you should note that interface does not have any constructors, only the methods. Because interface cannot be used to create any object, that is why no constructor is there in the interface.

Anyway, methods are there, now what are the methods? Many methods, I just highlights few methods and all the methods are have their brief description in the right side of the column. So if you want to know details about any methods, you should study the description, read the description those are provided and we will discuss all these methods and utilization with example programs and everything but not now in this video, in this lecture class. It will be discussed later on.

Now here we can see the first method. I will try to give a very, only few methods discussed in details but not all methods because it will really take enough time because lot of interfaces are to be covered. Now here the method, one method is called add method.

What is the meaning of this method? This method is, meanings that it has an argument of a template type, any type of objects. So if you want to add an integer to a collection integers then you can use it. So add method as the name implies it basically to add one element into a collection. Similarly, add all and we can see the argument is a collection, so if we pass a

collection to this method then all the elements which are there in this collection will be added to the existing collection.

If the addition is not possible then it will return false, if addition is successful it will return true, so that is why return concept is there. Now again let us see clear, as the method implies if we call the clear method for a collection let us say collection is x and if we call this method clear x dot clear, what will happen?

All the elements which are there in the collection x will be removed forever. So clear is one method. Now contains and argument is an object that means it will search, if a particular element obj is there in the existing collection or not. If it is present it will return true, if it is not present it returns false. ContainsAll, like contains only such but a collection, that means you give the input that a set of elements it will search the collection if those elements are there in the existing collection or not.

If it is there return true, if it is not there return false and here the collection that you passed and the collection it is there, not necessary the same sequence or same order. In any order, it will search one by one and then check if all presents there or not. So these are the methods are simply in a understandable form. So if you read the method and then description of each method, you will be able to understand what this method is doing.

(Refer Slide Time: 15:04)

Method	Description
<code>boolean equals(Object obj)</code>	Returns true if the invoking collection and <code>obj</code> are equal. Otherwise, returns false .
<code>int hashCode()</code>	Returns the hash code for the invoking collection.
<code>boolean isEmpty()</code>	Returns true if the invoking collection is empty. Otherwise, returns false .
<code>Iterator<T> iterator()</code>	Returns an iterator for the invoking collection.
<code>default Stream<E> parallelStream()</code>	Returns a stream that uses the invoking collection as its source for elements. If possible, the stream supports parallel operations.
<code>boolean remove(Object obj)</code>	Removes one instance of <code>obj</code> from the invoking collection. Returns true if the element was removed. Otherwise, returns false .
<code>boolean removeAll(Collection<?> c)</code>	Removes all elements of <code>c</code> from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false .
<code>default boolean removeIf(Predicate<? super T, P> p)</code>	Removes from the invoking collection those elements that satisfy the condition specified by <code>predicate</code> .
<code>boolean retainAll(Collection<?> c)</code>	Removes all elements from the invoking collection except those in <code>c</code> . Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false .

Table 7.2: The methods declared in Collection interface (continued...)

There are few more methods in this collection which I have mentioned here in this list. The equals method, equals method is basically check whether the existing collection and then the argument that we have passed or equals means if any object is present in this collection it is

basically there. Hash code, this concept we will discuss in details later on, basically it gives a code for a particular set or a particular element.

IsEmpty as the name implies is basically check whether collection presently contains any elements or it is empty. Iterator is the one method which we shall we use in details when we will discuss with programing examples that how to traverse a collection one by one, so is basically scanning or visiting the entire collection.

Now stream basically is a collection which will basically return a sequence of objects in the form of a stream. Stream is a one concept that is there in the Java and this basically returns stream. Remove is just like a clean but it is remove means a particular object if you want to remove from the existing collection.

RemoveAll whatever the elements it is there if it is present there it will remove all those elements. RemoveIf, if certain condition is satisfied then only it will remove so given under a predicate it will remove. RetainAll, collection C it will basically remove all the element expect those elements are mentioned in the inputs, so it is the method, so these are the methods are there.

Every methods has its own functionalities and those functionalities is basically better can be understood if we run a small program calling each method for a given collection then seeing the output, all those things we will do when we will discuss a specific data structure in our run actually. Now so these are the few methods.

(Refer Slide Time: 17:00)

Methods declared in Collection

Method	Description
<code>int size()</code>	Returns the number of elements held in the invoking collection.
<code>default Spliterator<E> spliterator()</code>	Returns a spliterator to the invoking collections.
<code>default Stream<E> stream()</code>	Returns a stream that uses the invoking collection as its source for elements. The stream is sequential.
<code>Object[] toArray()</code>	Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.
<code><T> T[] toArray(<u>array</u>[])</code>	Returns an array that contains the elements of the invoking collection. The array elements are copies of the collection elements. If the size of <i>array</i> equals the number of elements, these are returned in <i>array</i> . If the size of <i>array</i> is less than the number of elements, a new array of the necessary size is allocated and returned. If the size of <i>array</i> is greater than the number of elements, the array element following the last collection element is set to <i>null</i> and an error is reported.

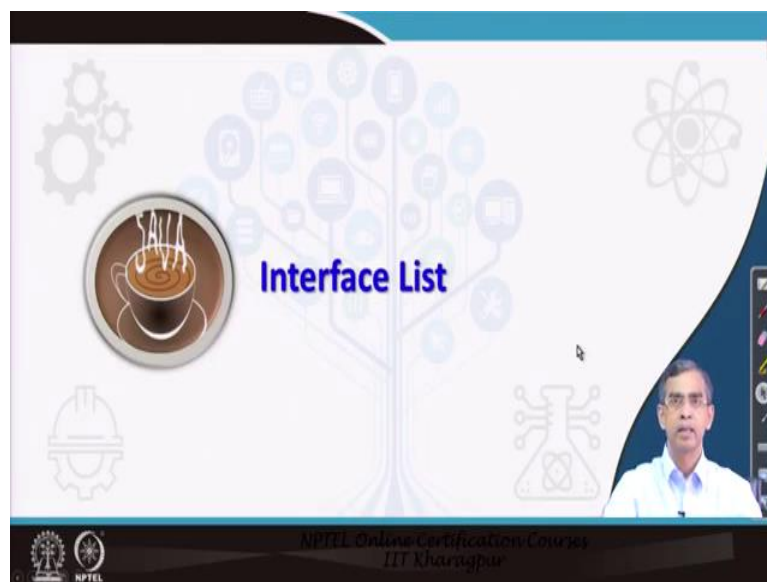
Table 7.2: The methods declared in Collection interface

NPTEL Online Certification Courses
IIT Kharagpur

There are few more methods in this interface which includes here. One important method that you can check it the iterator, split iterator like, stream is also another form of the stream. Now this method is very important, this method is called the bulk operation. What is the meaning? Meaning is that if you want to copy the existing collection into an array, so if you want to store the collection, now existing collection can be link list, can be tree, can be in the form of a other hash set or whatever it is there and it basically convert the existing collection whatever it is in the present form to store into an array form. Sometimes array is comfortable for many programmer, they want to have this collection in the form of an array, so it basically represent this one.

It is also same thing, it is basically the array means which collection you want to copy into another array. If you pass it, it will, this is basically a static method rule and it will work like. So these are the different methods those are there in collection and although methods should be implemented by certain classes corresponding to a particular data structure that these classes mean for. So this is the concept actually it is followed in Java collection framework composition.

(Refer Slide Time: 18:15)




Interface List

- The `List` interface extends `Collection` and declares the behavior of a collection that stores a **sequence of elements**. Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain **duplicate elements**.
- List is a generic interface that has this declaration:


```
interface List<T>
```

Here, T specifies the type of objects that the list will hold.

- In addition to the methods defined by `Collection`, `List` defines some of its own, which are summarized in Table 7.3.



Now let us come to the discussion of interface list, we will be able to see what are the different methods are declared there in the list interface. Interface list is basically another form of collection, another OA that a group of element will be stored is a list form. Now that is definitely different from others form, array form like this one.


Now it also mean for storing any type of collection, any type of elements so that is why template, so it is basically is an interface and this is the name of the interface list and the type of element that it can hold is basically template, that means it can store any type of data, integer, double floats, string or any user defined data it is there.

(Refer Slide Time: 19:18)

Methods declared in List

Method	Description
<code>void add(int index, E obj)</code>	Inserts <code>obj</code> into the invoking list at the index passed in <code>index</code> . Any preexisting elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
<code>boolean addAll(int index, Collection<? extends E> c)</code>	Inserts all elements of <code>c</code> into the invoking list at the index passed in <code>index</code> . Any preexisting elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns <code>true</code> if the invoking list changes and returns <code>false</code> otherwise.
<code>E get(int index)</code>	Returns the object stored at the specified index within the invoking collection.
<code>int indexOf(Object obj)</code>	Returns the index of the first instance of <code>obj</code> in the invoking list. If <code>obj</code> is not an element of the list, <code>-1</code> is returned.
<code>int lastIndexOf(Object obj)</code>	Returns the index of the last instance of <code>obj</code> in the invoking list. If <code>obj</code> is not an element of the list, <code>-1</code> is returned.
<code>ListIterator<E> listIterator()</code>	Returns an iterator to the start of the invoking list.

Table 7.3: The methods declared in List interface (continued...)



Now let us see what are the different methods are declared there in the list interface. It is basically very similar to the methods those are there, there are many name of the methods

will be very same as the collection because it is in way collection extends or list, basically list extends collection interface. So all the methods are there but it is basically overriding method because depending on the different structure the method are to be defined accordingly.

That is why name of the method may be same but the way how it can insert is totally different but this method how actually it insert it need not to be worried by the programmer. Programmer should not bother about it, programmer only should know that if I want to maintain a structure according to this form then I should call this method for this form. And it will work for you, that is all.

Now like this the add method as you see add, here one thing index that mean if you want to add one element so this is the element. In a particular position the index is a position. So it is index start from 0 to highest value. If it is not able to add it, it will not do anything but if it can add it, simply add it and it does not return anything, that is thing is there.

AddAll basically start index and then argument is their collection. So if you give the set of elements as an input and call this method and in index is another, so it will add all the elements at this location from there. And then get method is just opposite to add method it basically returns a particular object or elements which is present at a particular location, index is a location.

IndexOf object, so that means if you pass as an input an object and it will basically check the array, check the list and it basically gives in which location that object is present, that element is present, so indexOf basically says that if the element present in which location if presents are there. And last index Of as the name implies it basically in the list it will basically indicates in which location the last element is present.

And then object obj means the list may contains duplicate elements, so object obj indicates the last index of is basically the last occurrence of the elements.

(Refer Slide Time: 22:03)

Method	Description
ListIterator<E> listIterator(int index)	Returns an iterator to the invoking list that begins at the specified index.
E remove(int index)	Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one.
default void replaceAll(UnaryOperator<E> opToApply)	Updates each element in the list with the value obtained from the opToApply function.
E set(int index, E obj)	Assigns obj to the location specified by index within the invoking list. Returns the old value.
default void sort(Comparator<? super E> comp)	Sorts the list using the comparator specified by comp.
List<E> subList(int start, int end)	Returns a list that includes elements from start to end-1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

Table 7.3: The methods declared in List interface

NPTEL Online Certification Courses
IIT Kharagpur

So there are few more methods in the list which are again included here. So this method is basically iterator that means for travelling. Remove method is basically removing a particular elements from a particular location, it returns the elements removed.

And replaceAll it is basically indicates that all the elements of the current type will be replaced by a particular elements. Set is basically same thing as basically replace like but it is in particular elements, say particular object will be inserted actually and this is the one method the sort, if you want to sort all the elements in an order it is applicable for the numeric type data, so it is basically essentially is sorting method actually.

And subList is basically is a part of a list and then this part will starting from a particular location to end, so if the elements present there satisfying the start and end it will basically return a particular portion of that list. So this is the different methods those are there in the interface and we will be able to use all those methods when we will discuss list data structure, namely the link list data structure for example. So this is the list interface.

(Refer Slide Time: 23:25)

Interface Queue

NPTEL Online Certification Courses
IIT Kharagpur

Interfaces of collections

Interface	Description
Collection	Enables you to work with groups of objects; it is at the top of the collections hierarchy.
List	List extends Collection to handle sequences (lists) of objects.
Queue	Queue extends Collection to handle special types of lists in which elements are removed only from the head.
Deque	Deque extends Queue to handle a double-ended queue.
Set	Extends Collection to handle sets; which must contain unique elements.
SortedSet	Extends Set to handle sorted sets.
NavigableSet	NavigableSet extends SortedSet to handle retrieval of elements based on closest-match.

NPTEL Online Certification Courses
IIT Kharagpur

Likewise list, queue is the one, another structure. Queue is a specific or is a special data structure which allows only to insert one end and delete it another end, so it is called the insert add the rear position and delete from the front position. So there are two end actually, one is the front and another is the end.

So insertion will takes place at the rear position and deletion will takes place at the front position. So this concept is easier and then so this basically the in order to maintain a list there are different methods are declared and that methods are declared in the queue interface.

(Refer Slide Time: 24:19)

Interface Queue

- The `Queue` interface extends `Collection` and declares the behavior of a queue, which is often a first-in, first-out list.
- However, there are types of queues in which the ordering is based upon other criteria.
- `Queue` is a generic interface that has this declaration:

```
interface Queue<T>
```

Here, T specifies the type of objects that the queue will hold.
- The methods declared by `Queue` are shown in Table 7.4.

NPTEL Online Certification Courses
IIT Kharagpur

Now queue likewise other classes that we have, other interfaces that we have discussed it also allows you to store any type of objects, so that is why it is a template.

(Refer Slide Time: 24:33)

Methods declared in Queue

Method	Description
<code>element()</code>	Returns the element at the head of the queue. The element is not removed. It throws <code>NoSuchElementException</code> if the queue is empty.
<code>boolean offer(T obj)</code>	Attempts to add <code>obj</code> to the queue. Returns <code>true</code> if <code>obj</code> was added and <code>false</code> otherwise.
<code>T peek()</code>	Returns the element at the head of the queue. It returns <code>null</code> if the queue is empty. The element is not removed.
<code>T poll()</code>	Returns the element at the head of the queue, removing the element in the process. It returns <code>null</code> if the queue is empty.
<code>T remove()</code>	Removes the element at the head of the queue, returning the element in the process. It throws <code>NoSuchElementException</code> if the queue is empty.

Table 7.4: The methods declared in `Queue` interface

NPTEL Online Certification Courses
IIT Kharagpur

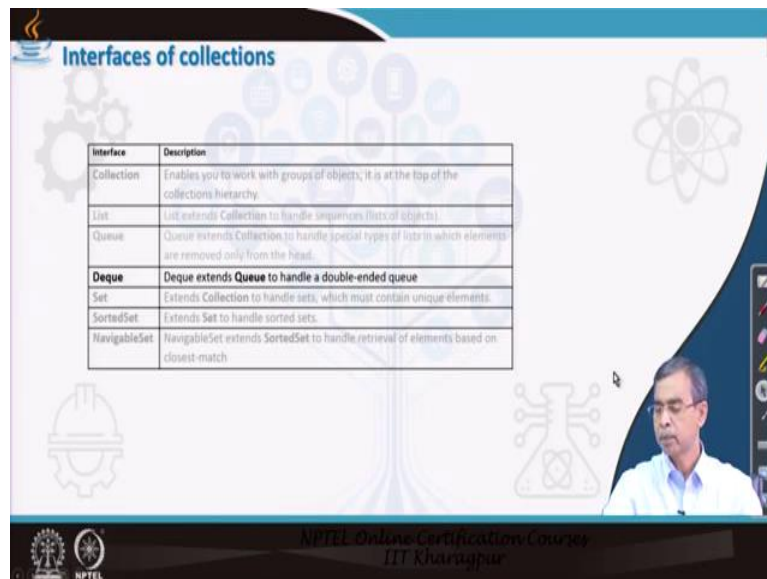
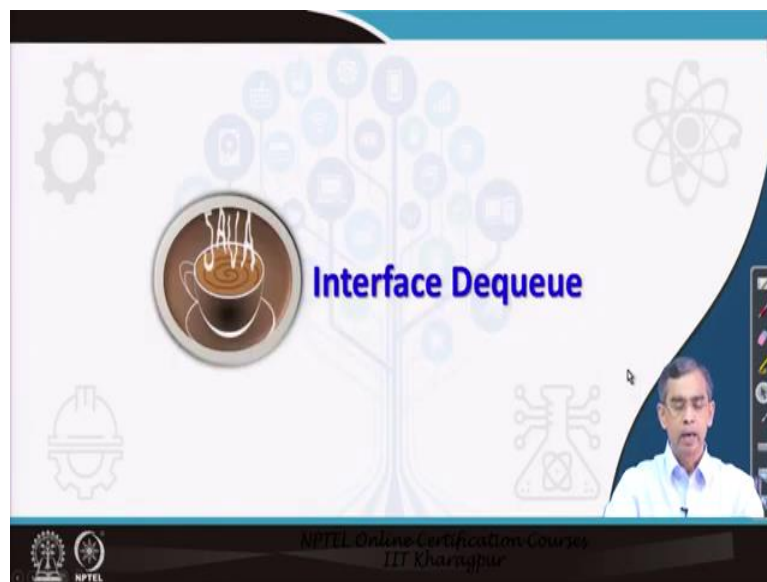
Now the methods those are there let us have a quick look of the methods, so this basically one method is called the elements, it basically return you the elements at the front of the queue. Offer is the one method which basically allow to add one elements into the queue.

Peek is the basically one element which will just return the element at the front but this element will just read not remove. Poll is the one element which basically remove as well as return the element which is at the front. And then remove is the one method, remove is

basically remove the head basically same as poll sort of thing, it will basically return the elements after removal.

And obviously, all those method return some exception if the queue is empty. So these are the methods are defined there in the interface queue.

(Refer Slide Time: 25:49)



And the de-queue is the another method. It is like queue, the difference is that it basically double ended. Insertion and deletion unlike queue can be done at any ends. For example in case of queue deletion is possible from one end, insertion is another end but here you can do at any end actually, so both the ends. That is why it is called the double ended queue, so this has more what is called the flexibility to add many insertion and deletion a person at the both end like.

(Refer Slide Time: 26:24)

Interface Dequeue

- The **Deque** interface extends **Queue** and declares the behavior of a double-ended queue.
- Double-ended queues can function as standard, **first-in, first-out queues** or as **last-in, first-out stacks**.
- **Deque** is a generic interface that has this declaration:

```
interface Deque<T>
```

Here, T specifies the type of objects that the deque will hold.
- In addition to the methods that it inherits from **Queue**, **Deque** adds those methods summarized in Table 7.5.

NPTEL Online Certification Courses
IIT Kharagpur

Now this are the, okay like queue other list, they also allow you to store any type of collection, so that is why the template.

(Refer Slide Time: 26:36)

Methods declared in Deueue

Method	Description
<code>void addFirst(E obj)</code>	Adds obj to the head of the deque. Throws an IllegalStateException if a capacity-restricted deque is out of space.
<code>void addLast(E obj)</code>	Adds obj to the tail of the deque. Throws an IllegalStateException if a capacity-restricted deque is out of space.
<code>Iterator<E> descendingIterator()</code>	Returns an iterator that moves from the tail to the head of the deque. In other words, it returns a reverse iterator.
<code>E getFirst()</code>	Returns the first element in the deque. The object is not removed from the deque. It throws NoSuchElementException if the deque is empty.
<code>E getLast()</code>	Returns the last element in the deque. The object is not removed from the deque. It throws NoSuchElementException if the deque is empty.
<code>boolean offerFirst(E obj)</code>	Attempts to add obj to the head of the deque. Returns true if obj was added and false otherwise. Therefore, this method returns false when an attempt is made to add obj to a full, capacity-restricted deque.
<code>boolean offerLast(E obj)</code>	Attempts to add obj to the tail of the deque. Returns true if obj was added and false otherwise.
<code>E peekFirst()</code>	Returns the element at the head of the deque. It returns null if the deque is empty. The object is not removed.

Table 7.5: The methods declared in Dequeue interface (continued...)

NPTEL Online Certification Courses
IIT Kharagpur

And these are many methods similar to the name of the method. All the methods are basically to add and remove or check whether it is empty or it basically return a part of the list and all these things are there. Only thing is that it specify some methods to add in either first, as a first element or last element or it is add any position all those things are there.

For example, addFirst, addLast and then getFirst, getLast. OfferFirst is basically adding again, offerLast. So peekFirst means the just read only not remove. So these are the different

methods are there like the other methods, it is basically for insertion, deletion and traverse, all these things are there.

(Refer Slide Time: 27:30)

Method	Description
E peekLast()	Returns the element at the tail of the deque. It returns null if the deque is empty. The object is not removed.
E pollFirst()	Returns the element at the head of the deque, removing the element in the process. It returns null if the deque is empty.
E pollLast()	Returns the element at the tail of the deque, removing the element in the process. It returns null if the deque is empty.
E pop()	Returns the element at the head of the deque, removing it in the process. It throws NoSuchElementException if the deque is empty.
void push(E obj)	Adds <i>obj</i> to the head of the deque. Throws an IllegalStateException if a capacity-restricted deque is out of space.
E removeFirst()	Returns the element at the head of the deque, removing the element in the process. It throws NoSuchElementException if the deque is empty.
boolean removeFirstOccurrence(Object obj)	Removes the first occurrence of <i>obj</i> from the deque. Returns true if successful and false if the deque did not contain <i>obj</i> .
E removeLast()	Returns the element at the tail of the deque, removing the element in the process. It throws NoSuchElementException if the deque is empty.
boolean removeLastOccurrence(Object obj)	Removes the last occurrence of <i>obj</i> from the deque. Returns true if successful and false if the deque did not contain <i>obj</i> .

Table 7.5: The methods declared in Dequeue interface

Now what I want to mention here is that all the interface depending upon the particular type of collection, the main operation those are required in order to maintain a particular collection is basically adding element, removing element, searching element, sorting element, traversing element and then check that whether that collection contain some elements or it is empty or this kind of, so those are called as status operation. And there are also some methods for the bulk operation is also there.

(Refer Slide Time: 27:58)

The slide features a central graphic of a coffee cup with steam rising from it, set within a circular frame. To the right of this graphic, the text "Interface Set" is displayed in a blue font. The background is light blue with a faint tree-like structure composed of various icons. A small inset video of a man in a white shirt is visible in the bottom right corner. The NPTEL logo and "NPTEL Online Certification Courses IIT Kharagpur" are at the bottom.

The slide is titled "Interfaces of collections" and contains a table with the following data:

Interface	Description
Collection	Enables you to work with groups of objects; it is at the top of the collections hierarchy.
List	List extends Collection to handle ordered lists of objects.
Queue	Queue extends Collection to handle special types of lists in which elements are removed only from the head.
Deque	Deque extends Queue to handle a double-ended queue.
Set	Extends Collection to handle sets, which must contain unique elements.
SortedSet	Extends Set to handle sorted sets.
NavigableSet	NavigableSet extends SortedSet to handle retrieval of elements based on closest-match.

The slide also features the same background graphics as the previous slide, including the coffee cup icon and the tree diagram. A small inset video of the same man is in the bottom right corner. The NPTEL logo and "NPTEL Online Certification Courses IIT Kharagpur" are at the bottom.

Now interface set is another group of collection which we will discuss in the next video.

(Refer Slide Time: 28:06)

Classes in Collection

NPTEL Online Certification Courses
IIT Kharagpur

Class Collection

NPTEL Online Certification Courses
IIT Kharagpur

Classes in collection

Class	Description
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of the List interface.
AbstractQueue	Extends AbstractCollection and implements parts of the Queue interface.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList .
ArrayList	Implements a dynamic array by extending AbstractList .
ArrayDeque	Implements a dynamic double-ended queue by extending AbstractCollection and implementing the Deque interface.
AbstractSet	Extends AbstractCollection and implements most of the Set interface.
EnumSet	Extends AbstractSet for use with enum elements.
HashSet	Extends AbstractSet for use with a hash table.
LinkedHashSet	Extends HashSet to allow insertion-order iterations.
PriorityQueue	Extends AbstractQueue to support a priority-based queue.
TreeSet	Implements a set stored in a tree. Extends AbstractSet .

Table 7.6: The classes derived **Collection** class

NPTEL Online Certification Courses
IIT Kharagpur

I just want to mention the different classes. As we have already mentioned these are the different classes are there, all these classes better can be understood while we discuss a corresponding data structure. For example, here Array list is the class to deal with array, Linklist is a class to deal with the link list concept in data structure, Priority queue is basically data structure related to the queue operation.

Array de-queue is basically representing the queue in the form of an array but restriction that insertion and deletion can be done only at the end not from the middle line. So these are the different what is called the classes are declared and we shall discuss all these classes as a data structure point of view and then their implementation and then how all this Java collection framework using Java collection framework they can be managed.

So these collections actually that we are going to discuss have many collection classes actually, all those classes therefore I can keep hold right now, we will not discuss at the moment.

(Refer Slide Time: 29:17)



Java data structures with collection

- You will learn how the different data structures that you can implement in your programs using the utility available in `java.util` package.
- Overall, all the data structures can be broadly classified into four categories. The broad data structures classification is shown in Table 7.9.

Data Structures	List	Queue	Set	Map
Indexed	ArrayList	ArrayDeque	HashSet	HashMap
Sequential	LinkedList	PriorityQueue	TreeSet	TreeMap
Indexed with links			LinkedHashSet	LinkedHashMap
Bit string			EnumSet	EnumMap

Table 7.7: Java Supports to data structures

NPTEL Online Certification Courses
IIT Kharagpur

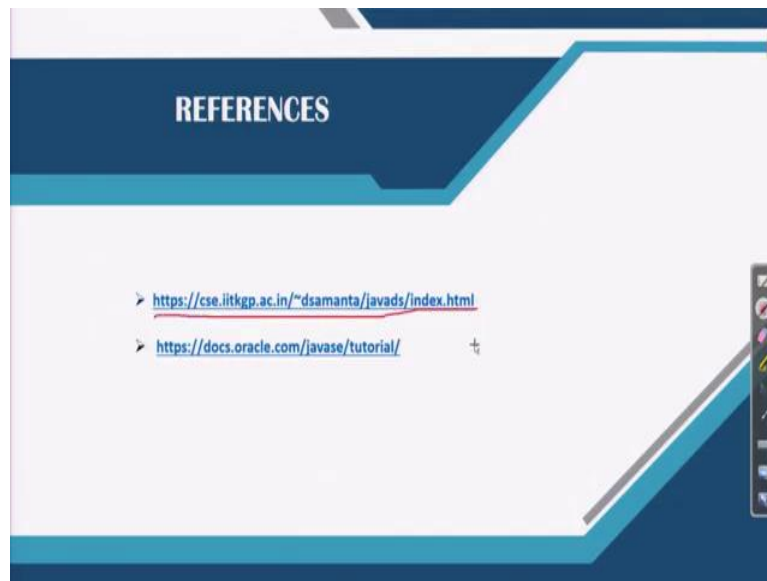
Now overall the different data structures those are suitable for different storing and retrieving they can be categorized into this kind of form either indexed or sequential. The ArrayList class which basically is an index type to maintain an array of elements, the LinkedList is a sequential type that means it is a sequential form, ArrayList is an indexed form.

Now the Array D-queue is again another indexed representation and PriorityQueue follow the sequential representation. So these are the different classes those are basically related to the data structure for certain type of collection.

And all those collection basically handled by corresponding classes in the, those are defined there in the collection framework. That means their methods and everything are defined, their methods or everything defined according to the declaration that we have learnt so far their interface is concerned.

For example, LinkedList basically implements all the methods those are there in the interface queue or interface list like. So these are the different classes that we will obviously discuss in details when a particular data structure will be discussed.

(Refer Slide Time: 30:53)



And regarding the detailed story about different structure and the different classes, different interfaces you can consult the Oracle documents, this is a tutorial form, it is very nice one but it is very exhaustive.

Usually it is good for advanced programmer. For the beginners those are new to this concept I should suggest them to consult this link, this link contains very easy and understandable manner of all the classes along with some example illustrating how the different methods defined in different classes can be used to perform certain operation.

However we shall discuss all these things in details throughout the course, so this is just a quick view of the collection framework, a part. Next part will be discussed in the lecture, this is regarding set. Set includes similar kind of interfaces and many classes there, their utilities and others, okay. Thank you.