Data Structures and Algorithms using Java

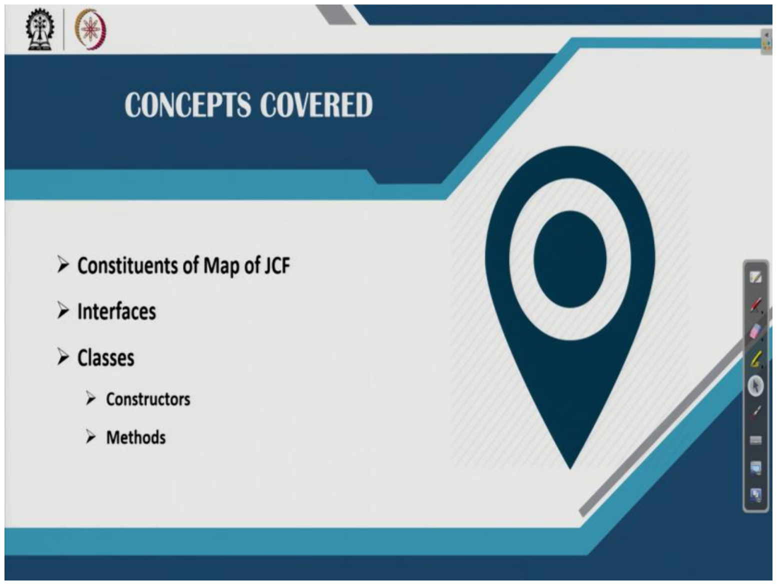
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Lecture 9

Map Framework

We are discussing about the collection and then Java supports to facilitate different operations on the collection. In the last two video lectures we have covered about the different views that a collection can be seen from the Java programming point of view or Java facilities point of view. There are many activities, many methods we have learned and altogether we call them as a collection.

(Refer Slide Time: 1:09)



CONCEPTS COVERED

- Constituents of Map of JCF
- Interfaces
- Classes
  - Constructors
  - Methods

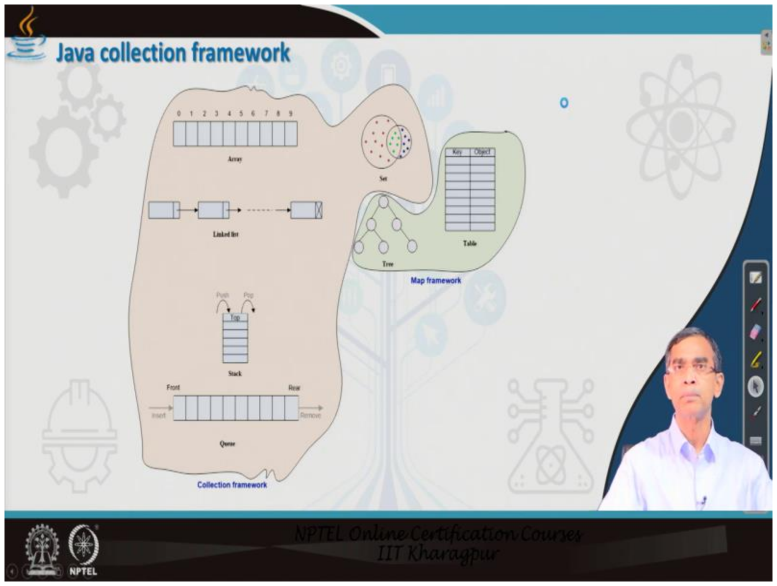
Today we will discuss another important framework. This is exactly not as a part of Java collection framework, it is a another framework, more precisely it is called map framework. This framework like Java collection framework is defined in java.util package. Now it has, it is a, actually it is a different stock of collection. Different stock of collection means it represents the objects or rather we can data structures from the different view point. This view point is actually called map.

(Refer Slide Time: 1:53)

Constituents of Map

Now in this lecture we will try to understand what are the composition of this map framework and what are the different classes and interfaces are there, so that map framework can be utilized by a programmer to develop his or her own application programs.

(Refer Slide Time: 2:23)



Java collection framework

Now before going to take the actual discussion, I would like to tell about in what the map framework is different than the collection framework that we have already studied. Now in the collection framework the data structure can be viewed either from indexed representation or sequential representation or linked representation whatever it is there. And there are five different structures that we have already studied namely arrays, linked list, stack and queue and set.

Now there are two more structural concepts in the theory of computer science that is there are two more data structures. They are called tree and table. Somehow we have some idea about tree based representation of a set. Actually this, in this sense tree structure is used to maintain a collection in the form of a set and in the collection in the form of a tree itself.

Now, anyway so this tree is basically considered a new what is called the collection and which is included in the another framework, the map framework. Other than this tree structure there is another structure, it is called the table. So table is also one sort of form, it is called the map framework. Now, here the question is arises that in what sense the two frameworks are different from each other, collection framework and map framework?

In this regard, I just want to mention one thing, in the collection that we have learned the objects are stored as a group of objects, they are stored either in array or in linked list or set, whatever it is. Whereas in map framework objects are stored, but objects are stored with two what is called the entities. One entity is basically the object itself that needs to be stored and another entity is called key of the objects.

What is the concept of key, that I can explain you with a simple example? Consider books, there is a number of books that needs to be stored in a programme suppose. So we can create a class book and accordingly create several objects of the type book. So book is a, book class can facilitate to create a collection of objects, they are called book objects.

Now fine, this is good idea, so this is idea so far collection framework maintains, but the map framework enhance the concept little bit in a other sense to facilitate a faster retrieval, storing as well as reading the data from the memory. So, faster storing and as well as faster retrieval. Now how it is possible? The idea it is basically incorporating key to each object. What is actually key?

For a book object, for example, the accession number of the book is basically is a key of the book. This means that if we give an input as an accession number then a corresponding book can be retrieved or if we want to store a book object, we can store both the information that is the accession number and the object itself.

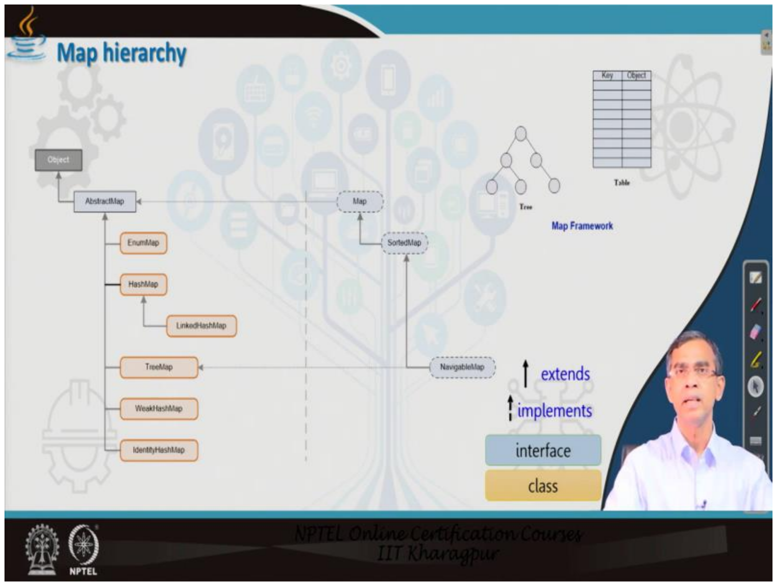
Likewise, student record, for example, another example, student record has many attributes like their name, their department, their date of joining, date of birth, marks obtained, different courses, like, like, so a record can be of using many data you need. Now in order to have a compact representation of the student, what we can use? We can use a number, let it be roll number.

Roll number can be anything like, anyway so what I can say is that objects along with a key, that means which is basically represents uniquely a particular objects can be used to store and retrieve in a an efficient way. Now again let us come to the key, key is what? As I said key is an identification number like accession number or roll number. A key value can be of any type again, it can be integer, roll number say 1, 2, 3, 4 like, it can be a string like accession number maybe A, then 2018 and B number say 4, 5.

So these are the strings, string is a basically alpha, is a collection of alpha numeric characters. Even it can be any objects again. So a key can be anything. Now what is the basic difference between the collection framework and map framework is that, while we are storing or retrieving an element into a collection or from a collection, we, in collection framework we only consider the object itself, object can be of any type.

Whereas, in case of map framework, we represents, we consider two entities, one is called the key and another the object itself, where key and entity both can be of any type. So this is the concept. Now based on this concept, the idea it is that how the map framework, Java developer has proposed.

(Refer Slide Time: 8:57)



So in this lecture we will try to have a quick understanding about the different in the map framework. Compare to collection framework, map framework is relatively simple, is a small, it has only three interfaces namely map, then sorted map and navigable map. So three interfaces are, in these three interfaces, the different methods are declared. So that all these methods, a Java programmer can use in their Programme to manipulate their collections or we can say their maps.

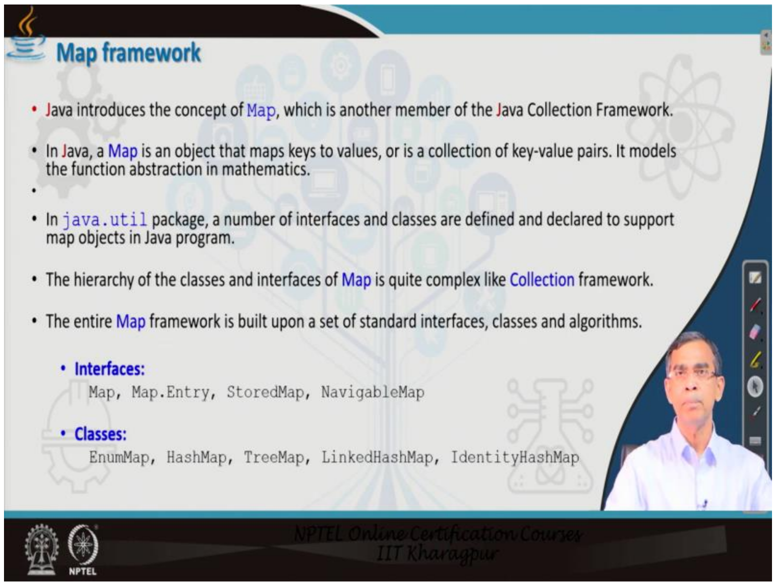
Now as you know all the interfaces needs to be implemented, so for the implementation of all those interfaces, Java supports in java.util several classes out of which abstract map is an abstract class. And there are several real classes are there like enum map, hashmap, linked hashmap, tree map and some less important, so far from the programmer's utility point of view are called weak hashmap and identity hashmap.

All these classes are inherited from the abstract class, abstract map. Abstract map implements map interface. Now, again in the interface hierarchy, sorted map is basically is inherited from the map and navigable map is basically inherited from the sorted map. And by virtue of inheritance as you saw whatever the methods those are there in map interface are also available in their sorted map or navigable map.

And in addition there are some of their own methods those are defined their interface. And again I repeat all classes basically implements all the methods, so methods are declared in interface and they are actually implemented in the class. Therefore, using all these class, you can create objects, objects of that particular class. For example, using hashmap you can create an object of hashmap.

Now an object of hashmap as you know, it is basically represents a collection and in this map framework, a collection consist of two pairs values, one is called the key and another is the value itself. So key and value, a group of keys and values constitute a collection in the map framework. So this is basically the fundamental what is called a concept, so far the map framework is concerned.

(Refer Slide Time: 11:50)



Map framework

- Java introduces the concept of Map, which is another member of the Java Collection Framework.
- In Java, a Map is an object that maps keys to values, or is a collection of key-value pairs. It models the function abstraction in mathematics.
-
- In `java.util` package, a number of interfaces and classes are defined and declared to support map objects in Java program.
- The hierarchy of the classes and interfaces of Map is quite complex like Collection framework.
- The entire Map framework is built upon a set of standard interfaces, classes and algorithms.
  - **Interfaces:**
    `Map, Map.Entry, StoredMap, NavigableMap`
  - **Classes:**
    `EnumMap, HashMap, TreeMap, LinkedHashMap, IdentityHashMap`

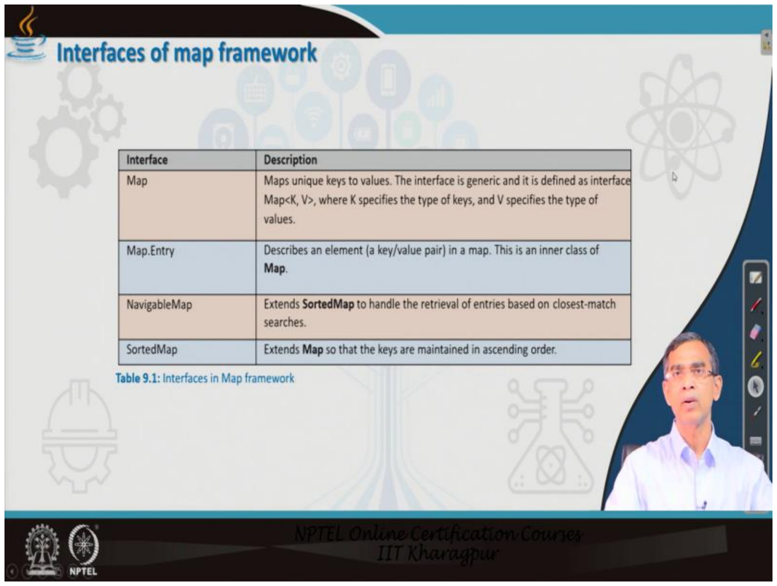
NPTEL Online Certification Courses
IIT Kharagpur

Now there are few interfaces we have already mentioned, like map, stored map and navigable map. There is one inner class it is basically defined within the map interface itself; it is called the map.Entry. Now, so these are the interface total that is there in the map framework and so far the classes are concerned as we have already mentioned enum map, hashmap, tree map, linked hashmap and weak hashmap and the identity hashmap are the classes those are defined there. And all these interfaces and classes are defined there in java.util package where collection framework is also defined there.

(Refer Slide Time: 12:37)

NPTEL Online Certification Courses
IIT Kharagpur

Interfaces in Map



NPTEL Online Certification Courses
IIT Kharagpur

Interfaces of map framework

Interface	Description
Map	Maps unique keys to values. The interface is generic and it is defined as interface Map<K, V>, where K specifies the type of keys, and V specifies the type of values.
Map.Entry	Describes an element (a key/value pair) in a map. This is an inner class of **Map**.
NavigableMap	Extends **SortedMap** to handle the retrieval of entries based on closest-match searches.
SortedMap	Extends **Map** so that the keys are maintained in ascending order.

Table 9.1: Interfaces in Map framework

Now we will quickly have a, quick round about the different interfaces those are there. First of all the map, it is basically the parent of interfaces belongs to this map framework and this map frameworks basically facilitates many methods which basically useful to manipulate any map of collection. Now entry is also is an inner class of this map, it also basically decides how an element, that means an element in map framework you can remember, it is basically key value pair.

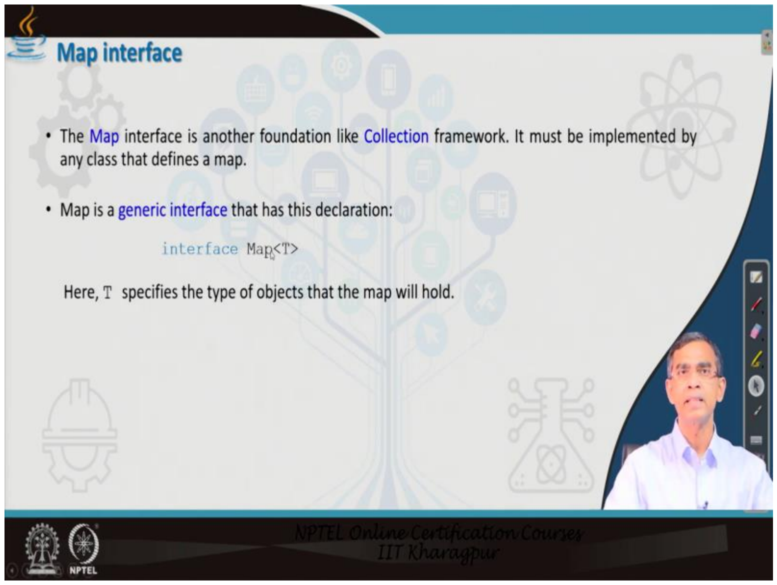
Unlike in collection only the value or object, here key and object pair in a map. So this is basically inner class so far map.Entry in concerned. Navigable map it is basically like map only but here in case of map, suppose you want to search a particular value or objects and then to search it you have to provide a key, if you do not provide key you will not be able to search, that is one important concept.

Even if you want to insert, first you decide a unique key, if the key is not unique then it is again a problem, whatever it be. So key has to be provided. Now in case of map, exactly if the key matches with the stored elements those are there, then fine, you can retrieve it. If you do not find, then navigable actually closest map, closest match, nearby this one it will say.

In some application it is required, exactly not the key value, but near about the key value, so that you can finally filter it and then get the actual value if you want from the filter value. And sorted map is same as map again, but the difference is that in case of map the elements that mean key value pair are not necessarily to be stored in a order, in an order, but in case of sorted map, all the objects are stored in the sorted order. Here sorted order is based on the key values.

(Refer Slide Time: 14:43)

Map Interface

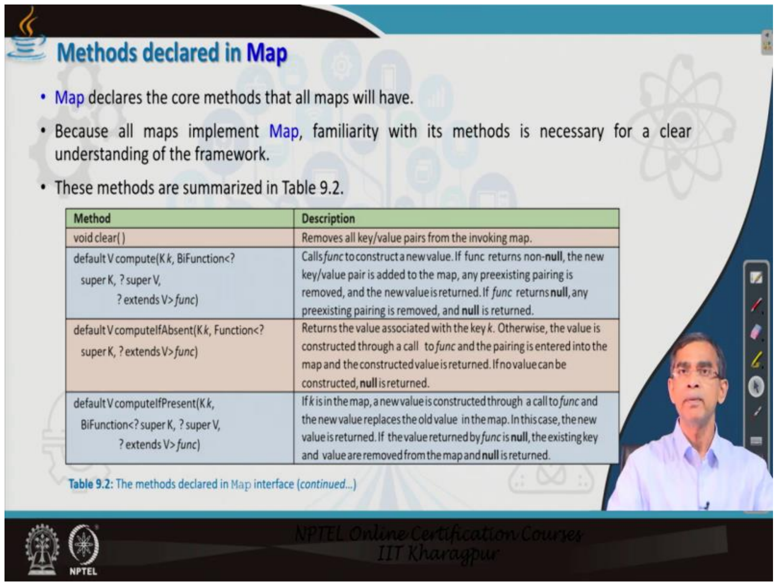


Map interface

- The Map interface is another foundation like Collection framework. It must be implemented by any class that defines a map.
- Map is a generic interface that has this declaration:

```
interface Map<T>
```

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



Methods declared in Map

- Map declares the core methods that all maps will have.
- Because all maps implement Map, familiarity with its methods is necessary for a clear understanding of the framework.
- These methods are summarized in Table 9.2.

Method	Description
void clear( )	Removes all key/value pairs from the invoking map.
default V compute(K *k*, BiFunction<? super K, ? super V, ? extends V> *func*)	Calls *func* to construct a new value. If func returns non-**null**, the new key/value pair is added to the map, any preexisting pairing is removed, and the new value is returned. If *func* returns **null**, any preexisting pairing is removed, and **null** is returned.
default V computeIfAbsent(K *k*, Function<? super K, ? extends V> *func*)	Returns the value associated with the key *k*. Otherwise, the value is constructed through a call to *func* and the pairing is entered into the map and the constructed value is returned. If no value can be constructed, **null** is returned.
default V computeIfPresent(K *k*, BiFunction<? super K, ? super V, ? extends V> *func*)	If *k* is in the map, a new value is constructed through a call to *func* and the new value replaces the old value in the map. In this case, the new value is returned. If the value returned by *func* is **null**, the existing key and value are removed from the map and **null** is returned.

Table 9.2: The methods declared in Map interface (*continued...*)

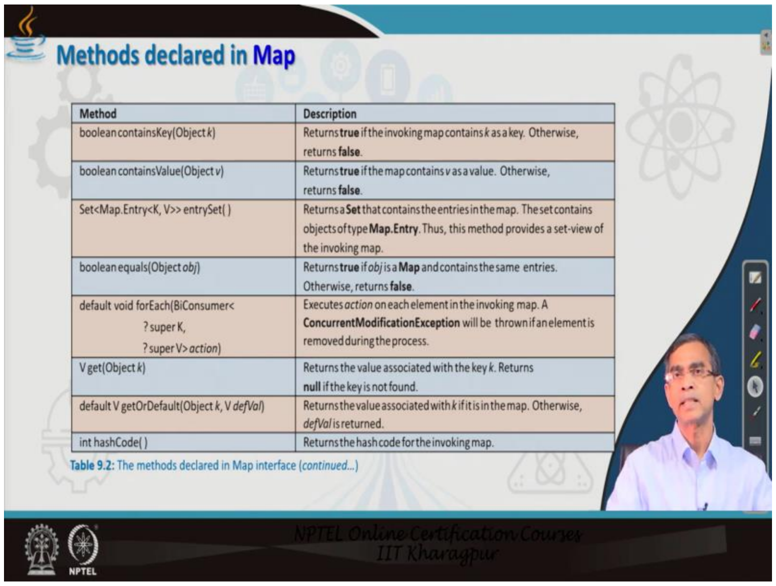
So these are the different maps interfaces and now we will quickly check what are the different methods are there in the map interface. Now map interface is a generic type, is basically represents this collection T and T specifies the object of this one. Now different methods those are there in the map framework I have listed in this tabular form that is for your future consultation so that you can go through this table and then learn each method separately.

As in this is a short lectures period, so it is not possible to discuss each and every method but here I just want to highlights about what exactly the methods about. All these methods basically gives support, gives basically the facilities in order to how a particular key value can be created, how key value and then object can be associated, how key value given a key value, how a particular object can be searched, how given a key value and object how it can be inserted, how given a key value and object how it can be removed and so many things are there.

Now for example here, in the method, that is the clear method. As the name implies we are already familiar to the clear method what it does for us in the context of collection. Here also clear method if we call for a collection that is a map collection, then it will clean all the entries those are there in the collection. So it basically, if you want to remove all the entries under the map collection then it will, you can use this method.

Now compute, here is basically the idea about how you can compute a key value for an object. So this method sometimes required for a given object if you want to generate the key values like this one. So there are different compute methods, compute present, compute absent, simple compute that means if a key value is present then how a new key value for a given object can be computed. If key value is absent, then what should be the methodology by which it can be computed and so many things are there.

(Refer Slide Time: 16:54)



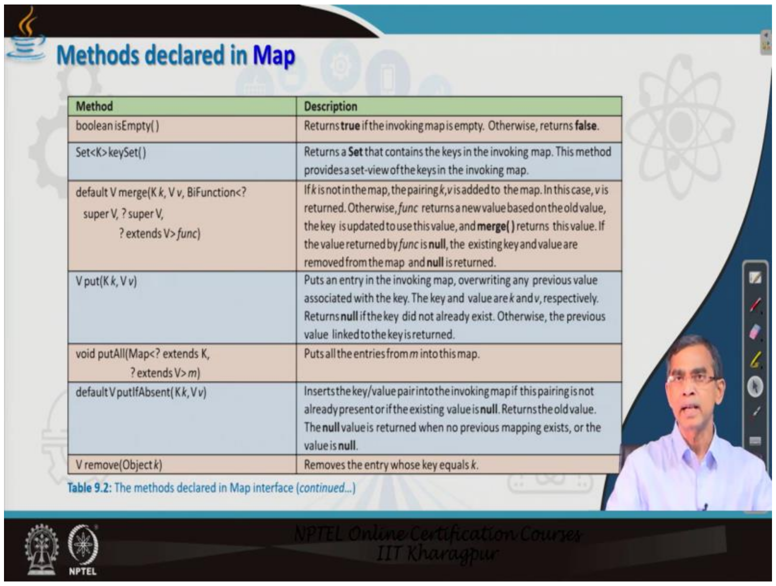
Methods declared in Map

Method	Description
boolean containsKey(Object *k*)	Returns **true** if the invoking map contains *k* as a key. Otherwise, returns **false**.
boolean containsValue(Object *v*)	Returns **true** if the map contains *v* as a value. Otherwise, returns **false**.
Set<Map.Entry<K, V>> entrySet( )	Returns a **Set** that contains the entries in the map. The set contains objects of type **Map.Entry**. Thus, this method provides a set-view of the invoking map.
boolean equals(Object *obj*)	Returns **true** if *obj* is a **Map** and contains the same entries. Otherwise, returns **false**.
default void forEach(BiConsumer< ? super K, ? super V> *action*)	Executes *action* on each element in the invoking map. A **ConcurrentModificationException** will be thrown if an element is removed during the process.
V get(Object *k*)	Returns the value associated with the key *k*. Returns **null** if the key is not found.
default V getOrDefault(Object *k*, V *defVal*)	Returns the value associated with *k* if it is in the map. Otherwise, *defVal* is returned.
int hashCode( )	Returns the hash code for the invoking map.

Table 9.2: The methods declared in Map interface (*continued...*)

Now in addition to this there are few, some methods are there which basically to know the status of the map collection, for example, the contents. Content means, should be particular object is present in the collection or not. And then the get method, get method means given an object what is the key value and vice versa, given a key value what is the object. And put value also it is there, equals method is also there, for example how to find if the two objects are same or not that belongs to this collection and likewise.

(Refer Slide Time: 17:32)



Methods declared in Map

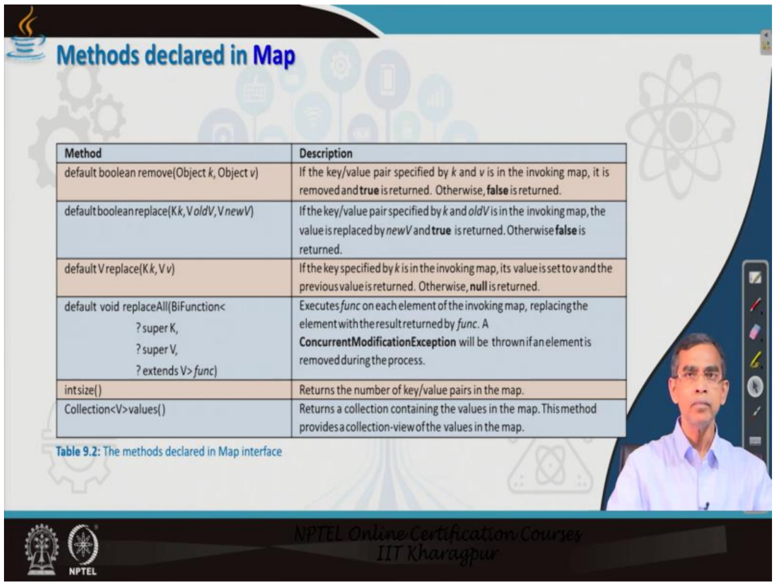
Method	Description
boolean isEmpty( )	Returns **true** if the invoking map is empty. Otherwise, returns **false**.
Set<K> keySet( )	Returns a **Set** that contains the keys in the invoking map. This method provides a set-view of the keys in the invoking map.
default V merge(K *k*, V *v*, BiFunction<? super V, ? super V, ? extends V> *func*)	If *k* is not in the map, the pairing *k*,*v* is added to the map. In this case, *v* is returned. Otherwise, *func* returns a new value based on the old value, the key is updated to use this value, and **merge( )** returns this value. If the value returned by *func* is **null**, the existing key and value are removed from the map and **null** is returned.
V put(K *k*, V *v*)	Puts an entry in the invoking map, overwriting any previous value associated with the key. The key and value are *k* and *v*, respectively. Returns **null** if the key did not already exist. Otherwise, the previous value linked to the key is returned.
void putAll(Map<? extends K, ? extends V> *m*)	Puts all the entries from *m* into this map.
default V putIfAbsent(K *k*, V *v*)	Inserts the key/value pair into the invoking map if this pairing is not already present or if the existing value is **null**. Returns the old value. The **null** value is returned when no previous mapping exists, or the value is **null**.
V remove(Object *k*)	Removes the entry whose key equals *k*.

Table 9.2: The methods declared in Map interface (*continued...*)

NPTEL Online Certification Courses
IIT Kharagpur

So there are many methods like this there are many methods to traverse the map collections also merging the two collection into a single collection using merge method. And whether a collection is empty or not, all these things can be obtained. If you want to remove a particular elements form the map collection, you can use the remove method from there. Put method is there, put conditionally, that means if a particular element is present in the collection already then if you want to insert it, how it can be done.

(Refer Slide Time: 18:02)



Methods declared in Map

Method	Description
default boolean remove(Object *k*, Object *v*)	If the key/value pair specified by *k* and *v* is in the invoking map, it is removed and **true** is returned. Otherwise, **false** is returned.
default boolean replace(K *k*, V *oldV*, V *newV*)	If the key/value pair specified by *k* and *oldV* is in the invoking map, the value is replaced by *newV* and **true** is returned. Otherwise **false** is returned.
default V replace(K *k*, V *v*)	If the key specified by *k* is in the invoking map, its value is set to *v* and the previous value is returned. Otherwise, **null** is returned.
default void replaceAll(BiFunction< ? super K, ? super V, ? extends V> *func*)	Executes *func* on each element of the invoking map, replacing the element with the result returned by *func*. A **ConcurrentModificationException** will be thrown if an element is removed during the process.
int size()	Returns the number of key/value pairs in the map.
Collection<V> values()	Returns a collection containing the values in the map. This method provides a collection-view of the values in the map.

Table 9.2: The methods declared in Map interface

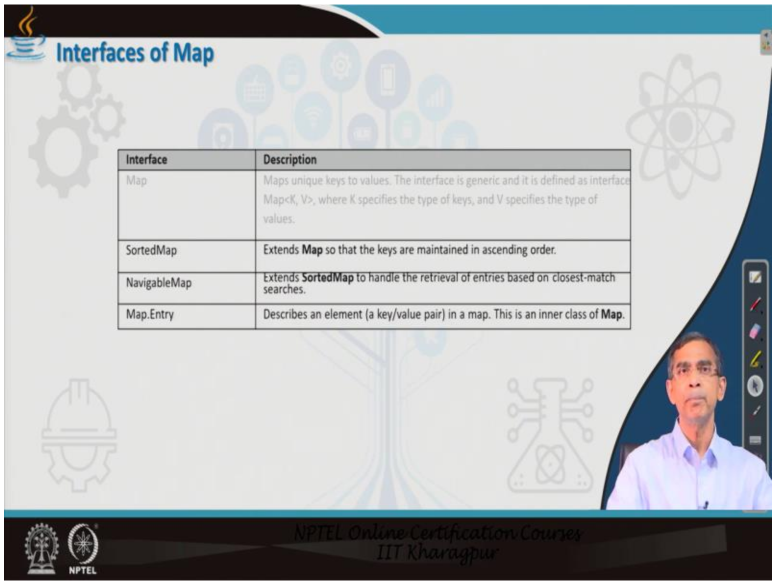
NPTEL Online Certification Courses
IIT Kharagpur

So there are so many facilities that is given by the Java developer and you can just simply as a fingertip use them in your Programme. Now regarding how to use all those methods, how to create collection, whether it is a simple collection or map collection we definitely study in due time and then you will be able to understand the strength of this java.util.

(Refer Slide Time: 18:24)

Interface SortedMap

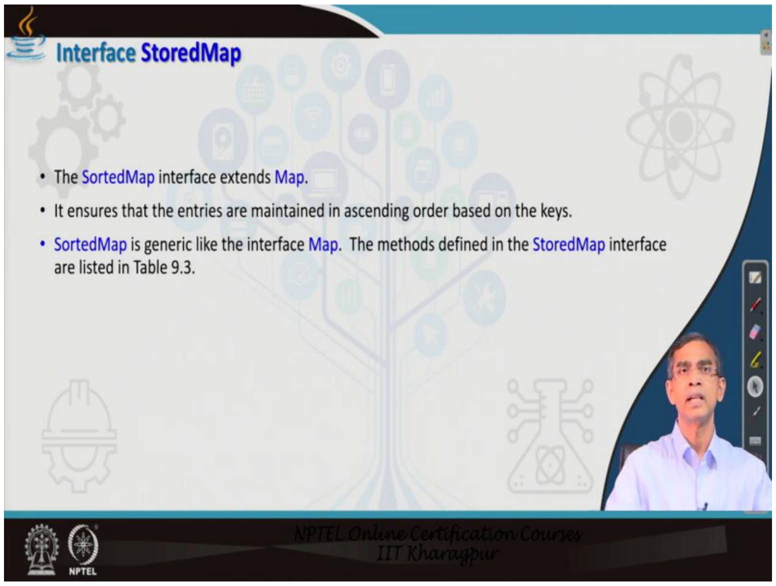
NPTEL Online Certification Courses
IIT Kharagpur



Interfaces of Map

Interface	Description
Map	Maps unique keys to values. The interface is generic and it is defined as interface Map<K, V>, where K specifies the type of keys, and V specifies the type of values.
SortedMap	Extends **Map** so that the keys are maintained in ascending order.
NavigableMap	Extends **SortedMap** to handle the retrieval of entries based on closest-match searches.
Map.Entry	Describes an element (a key/value pair) in a map. This is an inner class of **Map**.

NPTEL Online Certification Courses
IIT Kharagpur



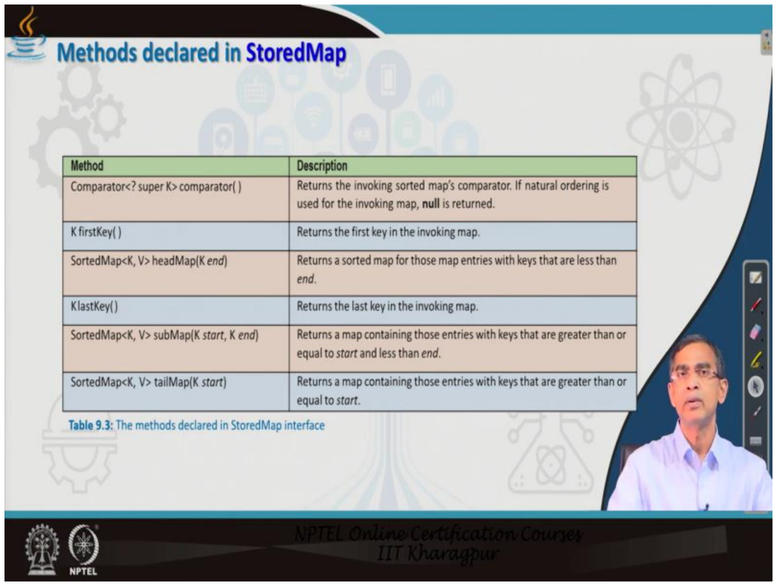
Interface StoredMap

- The SortedMap interface extends Map.
- It ensures that the entries are maintained in ascending order based on the keys.
- SortedMap is generic like the interface Map. The methods defined in the StoredMap interface are listed in Table 9.3.

NPTEL Online Certification Courses
IIT Kharagpur

Now this is the map framework and a map interface and you know map framework is the parent framework, parent interface and there is another is called a sorted map, which basically inherits the map framework itself. And as I told you the sorted map is similar to the map only but only it basically store the elements that is internally, you do not have to bother how to store. The Java compiler will take the enough what is called the facilities, enough what is called the step to add if you want to add to an existing collection a new object, then it will basically store them in a sorted manner.

(Refer Slide Time: 19:13)



Methods declared in StoredMap

Method	Description
Comparator<? super K> comparator( )	Returns the invoking sorted map's comparator. If natural ordering is used for the invoking map, **null** is returned.
K firstKey( )	Returns the first key in the invoking map.
SortedMap<K, V> headMap(K *end*)	Returns a sorted map for those map entries with keys that are less than *end*.
KlastKey()	Returns the last key in the invoking map.
SortedMap<K, V> subMap(K *start*, K *end*)	Returns a map containing those entries with keys that are greater than or equal to *start* and less than *end*.
SortedMap<K, V> tailMap(K *start*)	Returns a map containing those entries with keys that are greater than or equal to *start*.

Table 9.3: The methods declared in StoredMap interface

So basic idea of this sorted map interface is that faster retrieval, faster storing of the object compared to the simple map, using the map methods. Likewise, the methods in map, in stored map also there sorted map, there are many such methods available, they are, actually here it is a mistake, it is not the stored map, it should be sorted map.

Anyway, and so they are basically same method in addition to all the methods those are there in the map interface. Now some new methods like comparator that means if some objects needs to be consider which is not of numeric type is a usually specially defined, custom defined user type then it is your, it is the programmer's responsibility to define how two objects can be compared.

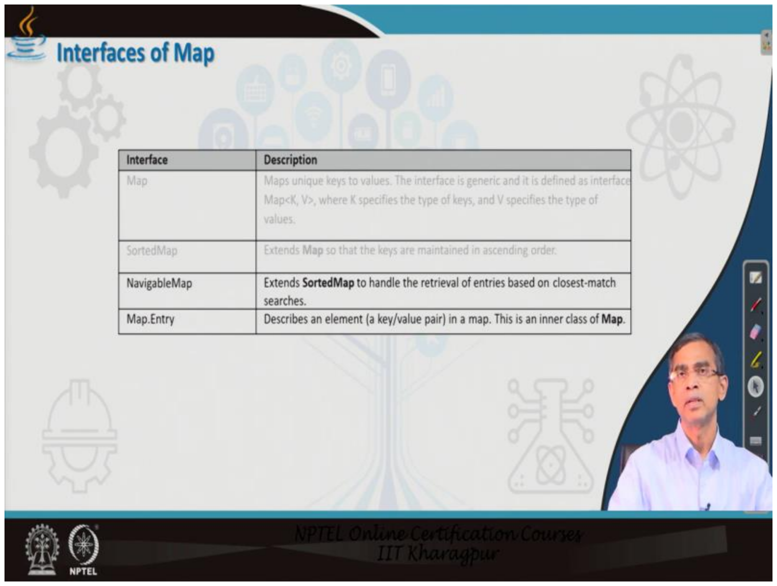
So compare to method we will discuss about how the method can be redefined for your own objects, so that means you have to, this is a method we can say that abstract method sort of thing or the method it is there a default and you can overwrite this method according to your need in your class. So, whenever you defined this class, you can create a compare to method which is an overwrite.

So you can extend, if you create a new Programme, you can just extend sorted map or you can extend hashmap or linked hashmap or T map, whatever it is there and then you redefine the compare to class there. Now, there is again few methods are there, those are special to this sorted map like first key as it represents that in the collection which is the starting key in the collection.

It returns the key object. Similarly, last key or just opposite to the first key and then it can tail map. Tail map means, if you can give a key value then corresponding to this key value, what are the next objects having key value after that one. So this key is a lower bound, then what is the upper key values that is there, so starting from the start values of the key. So these are the few special methods those are there in the sorted map interface.

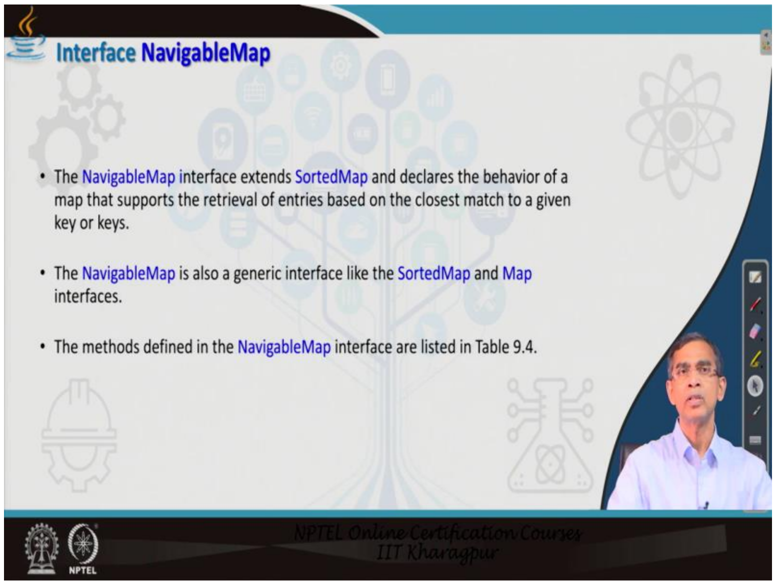
(Refer Slide Time: 21:20)

Interface NavigableMap



Interfaces of Map

Interface	Description
Map	Maps unique keys to values. The interface is generic and it is defined as interface Map<K, V>, where K specifies the type of keys, and V specifies the type of values.
SortedMap	Extends **Map** so that the keys are maintained in ascending order.
NavigableMap	Extends **SortedMap** to handle the retrieval of entries based on closest-match searches.
Map.Entry	Describes an element (a key/value pair) in a map. This is an inner class of **Map**.



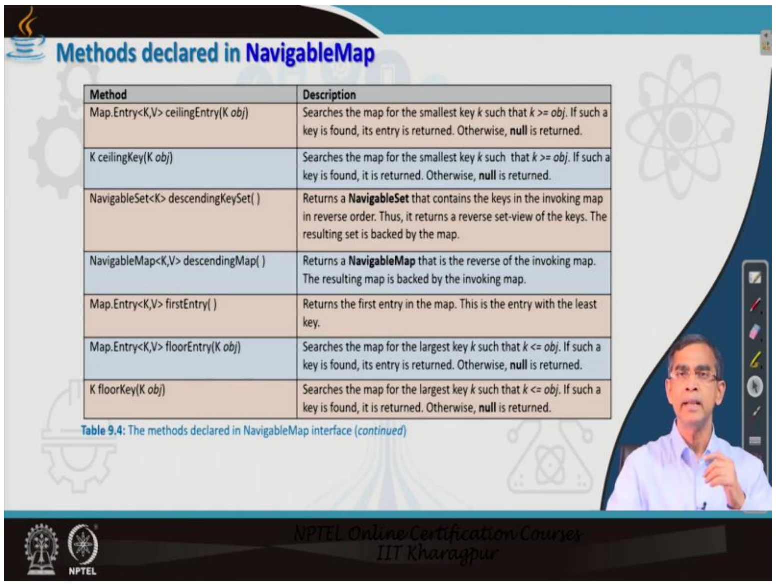
Interface NavigableMap

- The NavigableMap interface extends SortedMap and declares the behavior of a map that supports the retrieval of entries based on the closest match to a given key or keys.
- The NavigableMap is also a generic interface like the SortedMap and Map interfaces.
- The methods defined in the NavigableMap interface are listed in Table 9.4.

NPTEL Online Certification Courses
IIT Kharagpur

Now let us come to the discussion of interface navigable map. It is just like a map only but the idea is that here retrieval of entries based on the near about matches of the key values or the closest match that is based on the search. This is the only thing that it is the different here, it extends again the sorted map and declares the behavior of a map the supports for retrieval or addition or deleting, whatever it is there, same methods are there.

(Refer Slide Time: 22:08)



Methods declared in NavigableMap

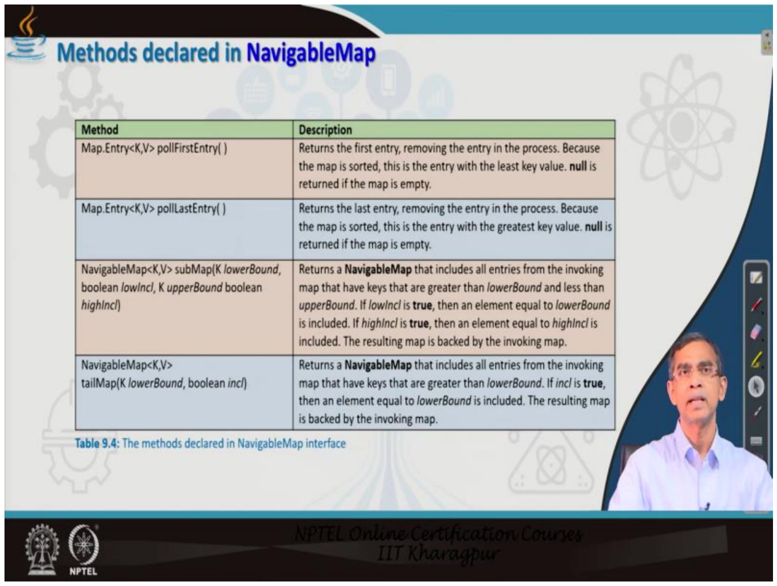
Method	Description
Map.Entry<K,V> ceilingEntry(K *obj*)	Searches the map for the smallest key *k* such that *k* >= *obj*. If such a key is found, its entry is returned. Otherwise, **null** is returned.
K ceilingKey(K *obj*)	Searches the map for the smallest key *k* such that *k* >= *obj*. If such a key is found, it is returned. Otherwise, **null** is returned.
NavigableSet<K> descendingKeySet( )	Returns a **NavigableSet** that contains the keys in the invoking map in reverse order. Thus, it returns a reverse set-view of the keys. The resulting set is backed by the map.
NavigableMap<K,V> descendingMap( )	Returns a **NavigableMap** that is the reverse of the invoking map. The resulting map is backed by the invoking map.
Map.Entry<K,V> firstEntry( )	Returns the first entry in the map. This is the entry with the least key.
Map.Entry<K,V> floorEntry(K *obj*)	Searches the map for the largest key *k* such that *k* <= *obj*. If such a key is found, its entry is returned. Otherwise, **null** is returned.
K floorKey(K *obj*)	Searches the map for the largest key *k* such that *k* <= *obj*. If such a key is found, it is returned. Otherwise, **null** is returned.

Table 9.4: The methods declared in NavigableMap interface (*continued*)

NPTEL Online Certification Courses
IIT Kharagpur

So there are few methods which are there in navigable map of it owns, here we have mentioned the methods. First of all ceiling entry, ceiling entry and floor entry are the two concepts that is used that okay what will be the highest values, what is the lowest value (concept) like and there is another method like descending, descending key set that means if you want to find in a descending order of the key values for a given key, you can use it and there is a first entry method, the floor entry method, the floor key method, these are the method basically related to key, how we can find its closest values to it.

(Refer Slide Time: 22:46)



Methods declared in NavigableMap

Method	Description
Map.Entry<K,V> pollFirstEntry( )	Returns the first entry, removing the entry in the process. Because the map is sorted, this is the entry with the least key value. **null** is returned if the map is empty.
Map.Entry<K,V> pollLastEntry( )	Returns the last entry, removing the entry in the process. Because the map is sorted, this is the entry with the greatest key value. **null** is returned if the map is empty.
NavigableMap<K,V> subMap(K *lowerBound*, boolean *lowIncl*, K *upperBound* boolean *highIncl*)	Returns a **NavigableMap** that includes all entries from the invoking map that have keys that are greater than *lowerBound* and less than *upperBound*. If *lowIncl* is **true**, then an element equal to *lowerBound* is included. If *highIncl* is **true**, then an element equal to *highIncl* is included. The resulting map is backed by the invoking map.
NavigableMap<K,V> tailMap(K *lowerBound*, boolean *incl*)	Returns a **NavigableMap** that includes all entries from the invoking map that have keys that are greater than *lowerBound*. If *incl* is **true**, then an element equal to *lowerBound* is included. The resulting map is backed by the invoking map.

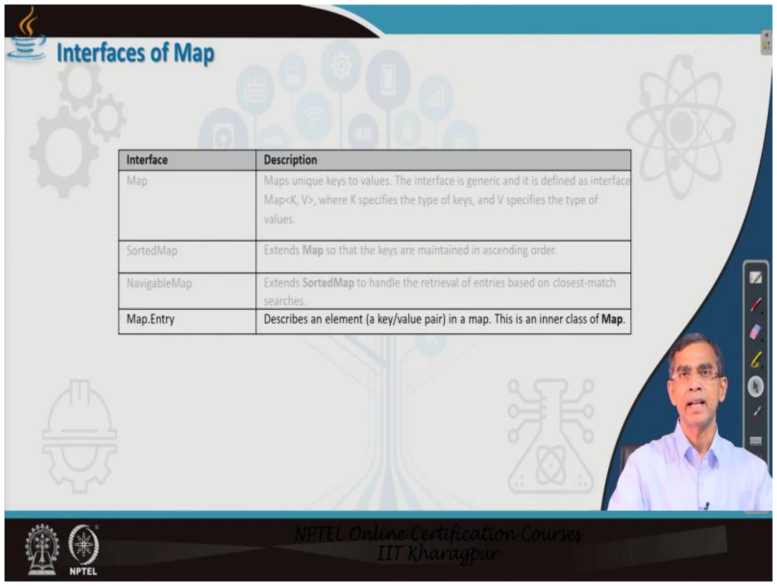
Table 9.4: The methods declared in NavigableMap interface

Then higher key, the lower key and then there is a navigable key, those are the methods. Higher entry methods and then headmap method, all these methods are basically the different way that the key and the value object pair can be managed, can be handled. So that is why the different methods have been incorporated in the navigable map interface.

(Refer Slide Time: 23:11)

Interface Map.Entry

NPTEL Online Certification Courses
IIT Kharagpur



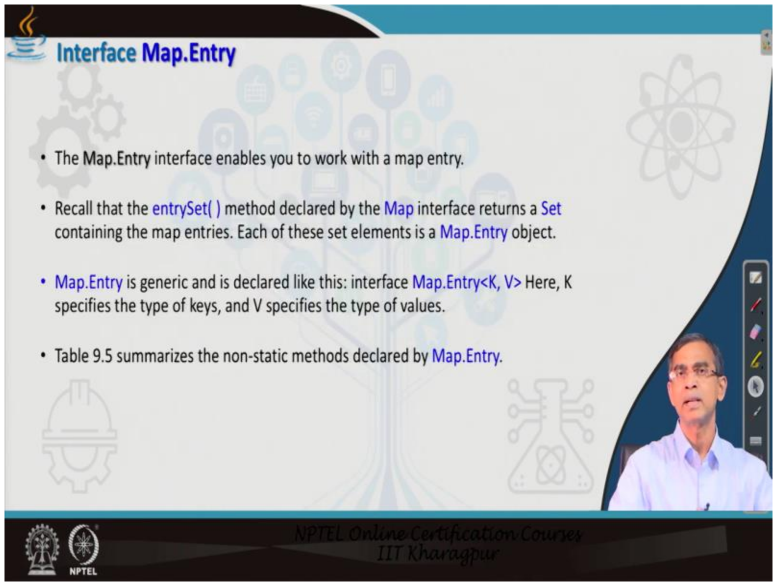
Interfaces of Map

Interface	Description
Map	Maps unique keys to values. The interface is generic and it is defined as interface Map<K, V>, where K specifies the type of keys, and V specifies the type of values.
SortedMap	Extends **Map** so that the keys are maintained in ascending order.
NavigableMap	Extends **SortedMap** to handle the retrieval of entries based on closest-match searches.
Map.Entry	Describes an element (a key/value pair) in a map. This is an inner class of **Map**.

NPTEL Online Certification Courses
IIT Kharagpur

And then map dot entry is a inner class, it does not have much special about but only it consider that how a key value pair can be enter into a map.

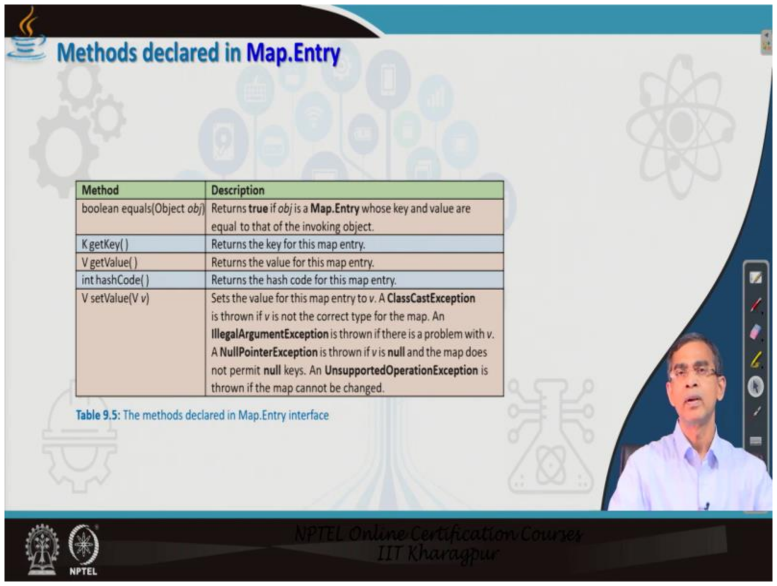
(Refer Slide Time: 23:20)



Interface Map.Entry

- The **Map.Entry** interface enables you to work with a map entry.
- Recall that the entrySet( ) method declared by the Map interface returns a Set containing the map entries. Each of these set elements is a Map.Entry object.
- Map.Entry is generic and is declared like this: interface Map.Entry<K, V> Here, K specifies the type of keys, and V specifies the type of values.
- Table 9.5 summarizes the non-static methods declared by Map.Entry.

NPTEL Online Certification Courses
IIT Kharagpur



Methods declared in Map.Entry

Method	Description
boolean equals(Object *obj*)	Returns **true** if *obj* is a **Map.Entry** whose key and value are equal to that of the invoking object.
K getKey( )	Returns the key for this map entry.
V getValue( )	Returns the value for this map entry.
int hashCode( )	Returns the hash code for this map entry.
V setValue(V *v*)	Sets the value for this map entry to *v*. A **ClassCastException** is thrown if *v* is not the correct type for the map. An **IllegalArgumentException** is thrown if there is a problem with *v*. A **NullPointerException** is thrown if *v* is **null** and the map does not permit **null** keys. An **UnsupportedOperationException** is thrown if the map cannot be changed.

Table 9.5: The methods declared in Map.Entry interface

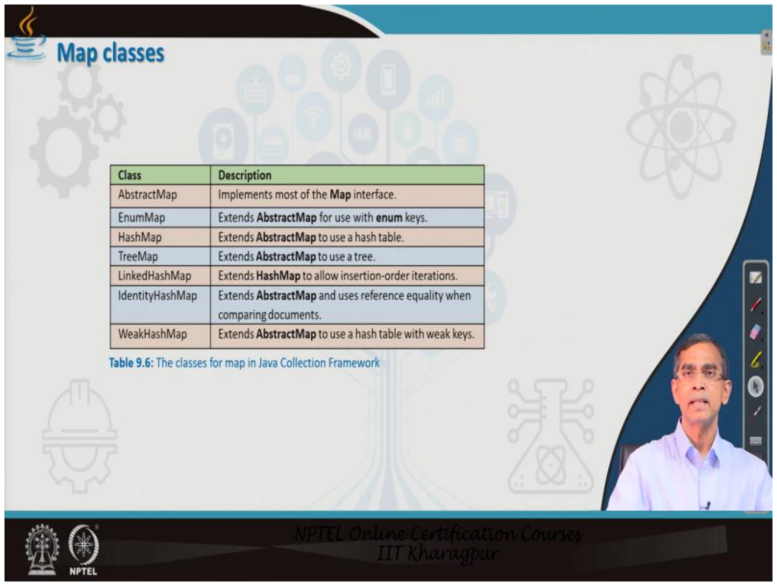
NPTEL Online Certification Courses
IIT Kharagpur

So it does not so much have its own much method but only few methods are there get key, get value, hash code create, set value create and then whether the two values are equals or not, all this things are there. So it is basically some peculiar reason of having a limited scope for the data security, the map dot entry interface has been introduced.

(Refer Slide Time: 23:46)

Map Classes

NPTEL Online Certification Courses
IIT Kharagpur



Map classes

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.

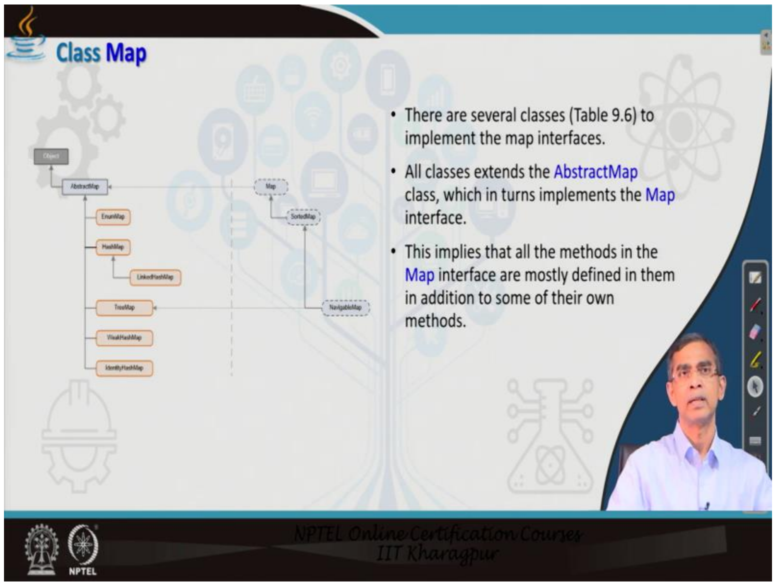
Table 9.6: The classes for map in Java Collection Framework

NPTEL Online Certification Courses
IIT Kharagpur

Now let us come to the classes. There are few classes, we have already mentioned, apart from this abstract map, the other important classes those are there is enum map, the hashmap, tree map and linked hashmap. So mainly these are the (three) four, four classes are very crucial and useful so far the map framework dealing is concerned. So as a programmer you have a very good knowledge about all these four classes.

Now again I repeat all these four classes are basically the implementation of the interfaces that we have just now discussed like map, that sorted map and then navigable map, whatever it is. Now this means that all the methods that we have discussed so far interface is concerned, they are implemented, they are in the classes, in addition some classes have their own methods also. So if I do not mention any methods, which already we have discussed about interface, it implies that these methods by virtue of implementation are there.

(Refer Slide Time: 24:59)



Class Map

- There are several classes (Table 9.6) to implement the map interfaces.
- All classes extends the AbstractMap class, which in turns implements the Map interface.
- This implies that all the methods in the Map interface are mostly defined in them in addition to some of their own methods.

Now we will discuss about which are the different methods are there in the, in their own classes actually.

(Refer Slide Time: 25:07)

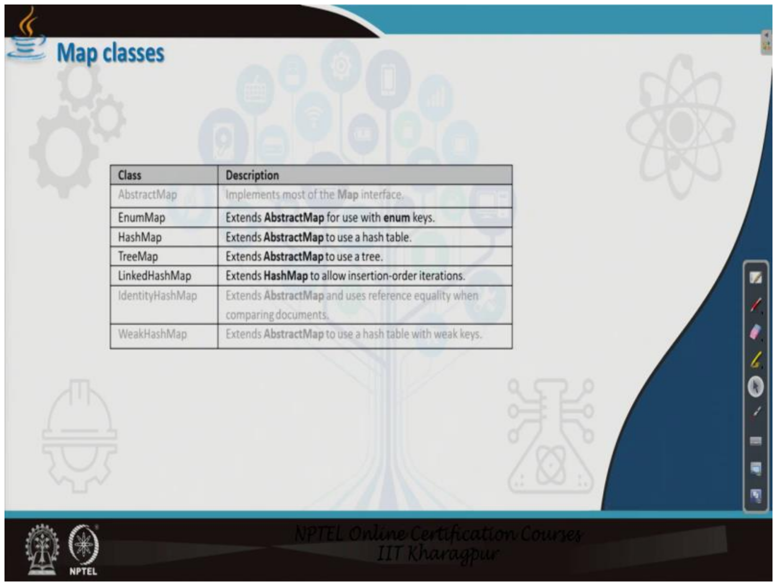
EnumMap Class

So we will discuss about, let us start about the enum map. Now I just want to say enum is basically is a kind of object. Like integer is a kind of object, float is a kind of object, enum is a kind of object. This basically is the, enum is a type of object I can say. Now what is a type of object? It is basically enumerated that means a particular value you can fix, just like enumerated value. For example, an object say Sunday, another object is Monday, another object is Tuesday, so these are the basically enumerated object.

I can say day is a enum type and in this enum type what are the different values that is possible? Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, like this, so seven what is called objects. So enumerated is basically just like a set and it has only certain range of values which is possible for the type. So enumerated is basically is a special type and it obviously all the elements those are there in the range are all unique.

So in that case the enumerated is closely resemble with set type. Now here so map is concern, each object should be associated with the key value. So Sunday is an object, its key value maybe I can give it 0, Monday is another object I can give the key value 1, Tuesday is another object I can give key value 3, so each key values is there. Now if we want to create a large collection of type enum, then definitely for the fastest or efficient mechanism you should use both key values as well as the enumerated values.

(Refer Slide Time: 26:49)



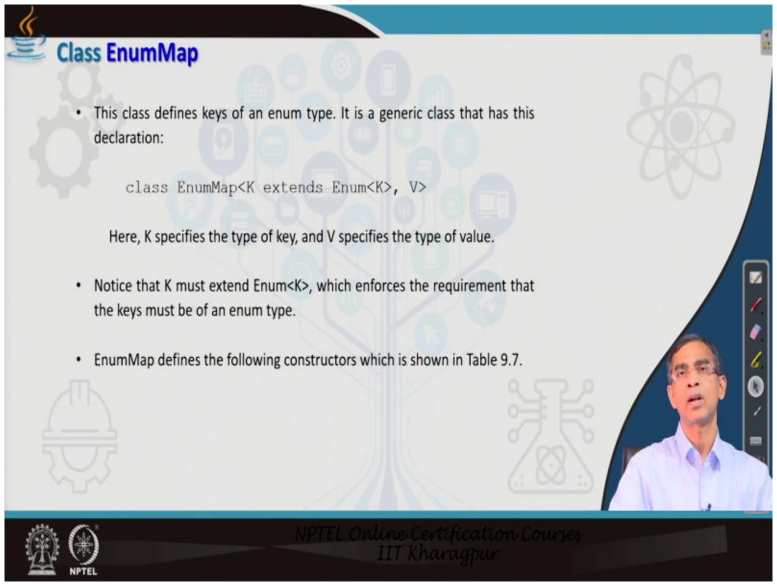
Map classes

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.

NPTEL Online Certification Courses
IIT Kharagpur

Now this way this enum map is basically it is there and as we can say that it basically extend abstract map for use with enum keys.

(Refer Slide Time: 27:00)



Class EnumMap

- This class defines keys of an enum type. It is a generic class that has this declaration:

```
class EnumMap<K extends Enum<K>, V>
```

Here, K specifies the type of key, and V specifies the type of value.

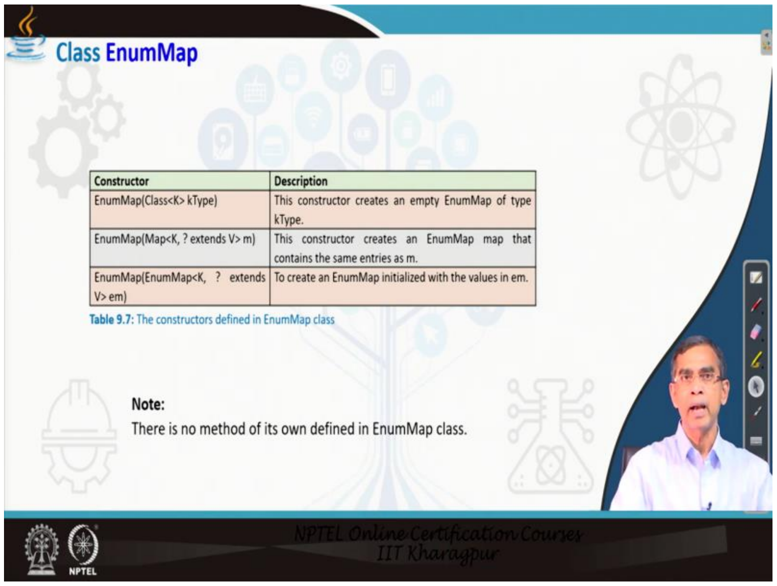
- Notice that K must extend Enum<K>, which enforces the requirement that the keys must be of an enum type.
- EnumMap defines the following constructors which is shown in Table 9.7.

NPTEL Online Certification Courses
IIT Kharagpur

So here key is basically the enumerated type and the object can be of any type like. Now, again enum map is a class, so this means it has two elements, one is called the K, K extends enum, that means K always should be of enumerated type. Beyond any enumerated type, it cannot be, that mean K cannot be a number or it is string like. So this is the specialty so far this key values is concerned because I told that key can be of any type.

This means that any type means number, or string or any user's defined type but for this enum map specialty, the key value should be of enumerated type. And then V is the object, that is value is actually which you want store. There means a key value should be enumerated type by which a particular objects can be associated and if we want to access a particular object, then we should access through its enumerated key.

(Refer Slide Time: 27:57)



Class EnumMap

Constructor	Description
EnumMap(Class<K> kType)	This constructor creates an empty EnumMap of type kType.
EnumMap(Map<K, ? extends V> m)	This constructor creates an EnumMap map that contains the same entries as m.
EnumMap(EnumMap<K, ? extends V> em)	To create an EnumMap initialized with the values in em.

Table 9.7: The constructors defined in EnumMap class

Note:
There is no method of its own defined in EnumMap class.

NPTEL Online Certification Courses
IIT Kharagpur

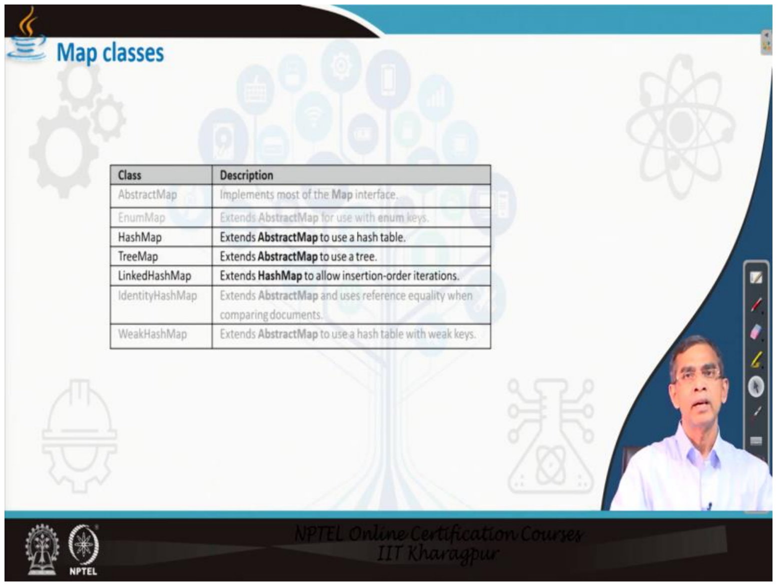
And it has only few methods, these constructors, only three constructors. The first constructor as you see here the enum, this is a very simple constructor to create a collection, what we have to, we have to give a input is that, what is the type of key values and key value itself. So by this way an object can be created. And the second constructor, basically it gives both key values and then the value itself. So key is always extends the map key and extends what is called the way. And this is basically you know this a bound is there, upper bound.

So that means it is basically applicable map object only. And then, this is another, where we can give the extension, it is the same way, this basically map K and it is basically is upper bound extends V and it is a enumerated map. So these are the different way, we will see exactly with an example, how the different objects can be created using this map. And yes one thing is that, the enum map class, it does not have any exclusive method of its own.

(Refer Slide Time: 29:13)

HashMap Class

NPTEL Online Certification Courses
IIT Kharagpur

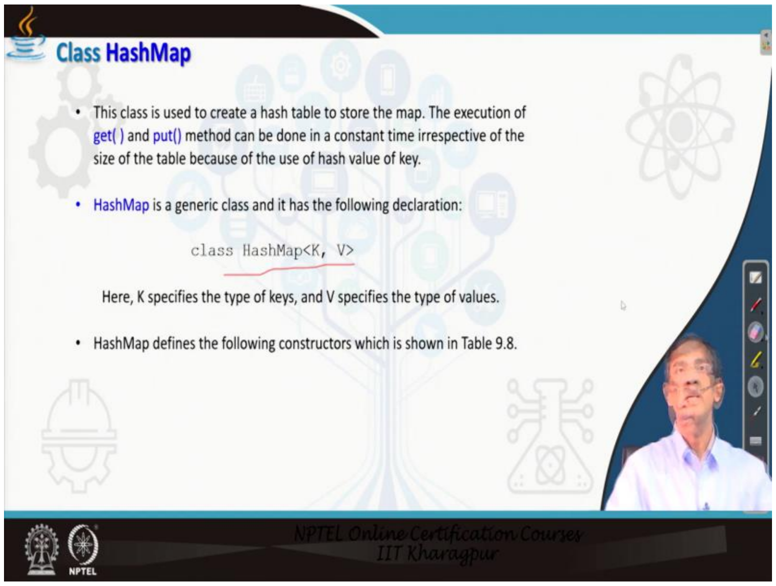


Map classes

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.

Now next let us discuss about hashmap class. Hashmap class again is a child class of abstract map, this means that whatever the methods those are declared there in map is also accessible to hashmap in addition to its own methods.

(Refer Slide Time: 29:33)



Class HashMap

- This class is used to create a hash table to store the map. The execution of get( ) and put() method can be done in a constant time irrespective of the size of the table because of the use of hash value of key.
- HashMap is a generic class and it has the following declaration:

```
class HashMap<K, V>
```

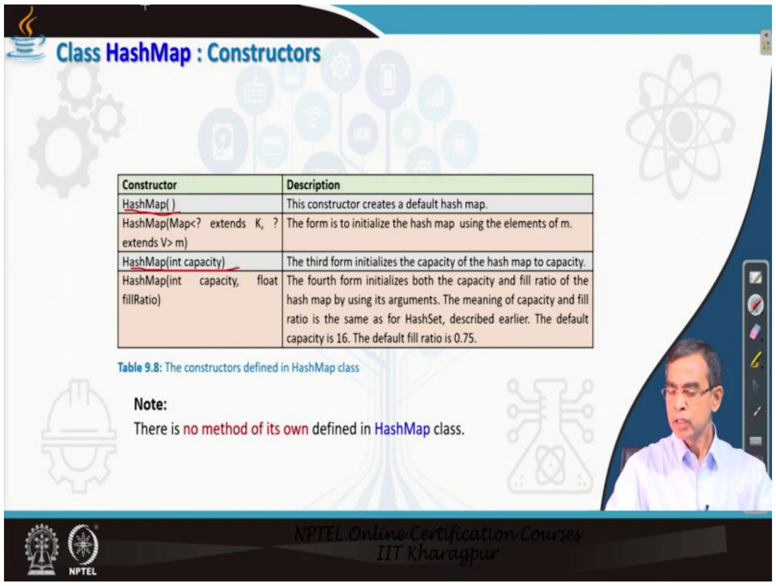
Here, K specifies the type of keys, and V specifies the type of values.

- HashMap defines the following constructors which is shown in Table 9.8.

NPTEL Online Certification Courses
IIT Kharagpur

Now let us see what are different way that hashmap collection can be created, as I already mentioned that it is not a single pair, it is basically a single value. The hashmap should be key and then the object value itself. So this is the composition that the collection should have.

(Refer Slide Time: 29:52)



Class HashMap : Constructors

Constructor	Description
HashMap( )	This constructor creates a default hash map.
HashMap(Map<? extends K, ? extends V> m)	The form is to initialize the hash map using the elements of m.
HashMap(int capacity)	The third form initializes the capacity of the hash map to capacity.
HashMap(int capacity, float fillRatio)	The fourth form initializes both the capacity and fill ratio of the hash map by using its arguments. The meaning of capacity and fill ratio is the same as for HashSet, described earlier. The default capacity is 16. The default fill ratio is 0.75.

Table 9.8: The constructors defined in HashMap class

Note:
There is no method of its own defined in HashMap class.

NPTEL Online Certification Courses
IIT Kharagpur

Now the constructors, those are there in hashmap, they are pretty simple, it is just similar to just this one. So, there hashmap this is default constructor that we can have. This is the default constructor, we do not have to pass anything but sometimes you can pass many information to it. For example, hashmap in capacity, so initially when you are to create a map collection, you can mention what will be the size. It is for basically efficient memory management, if you say that okay my collection have only size 100, you can mention it.

However, the initial capacity that you mention and if later on if you found that it require 1000 entries to be maintained then absolutely it is not an issue because the size of the map collection automatically grow itself depending on the integer there. Now for this automatic or dynamic progression of this map collection you can better for the memory management, if you know in which increment that the memory should grow.

So in order to do these things, you can mention the fill ratio. For example, 75 percent, if the 75 percent is filled then you can go for increasing into another 10 size or these kind of thigs are there. So default capacity is 16th but if you do not mention, for example different constructor type you use then a map will be created with a default capacity as 16.

But if you can mention the flip ratio or say 0.5 or 0.8 or whatever it is there, it basically, automatically grow these things. So these are the few constructors those are there in order to create the map collection for your application. There are certain methods, for the hashmap also there is no exclusive method other than the method those are there in map interface.

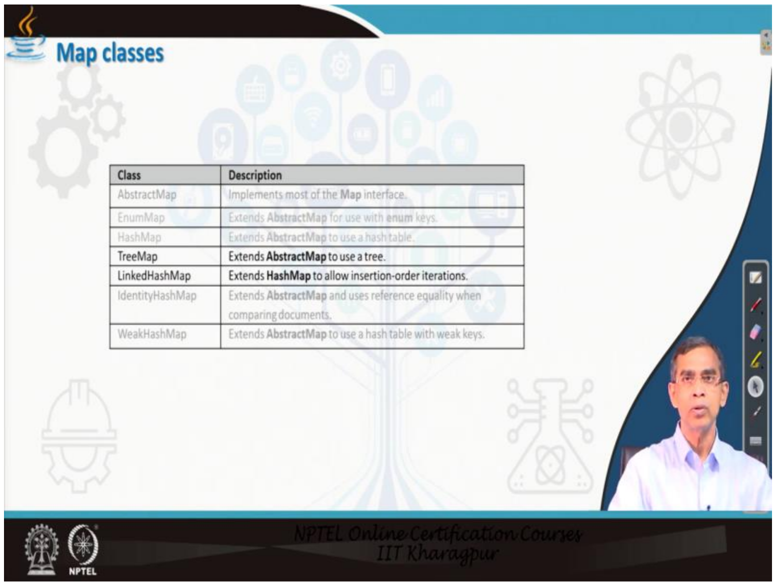
(Refer Slide Time: 31:54)

TreeMap Class

Now let us come to the tree map. It is basically same as the hashmap, only the difference, the difference is from the internal point of view, not that programmer's point of view. From the internal point of view is that hashmap is basically stored using indexed representation, whereas the tree map is used to store the data in linked representation.

Now there are three representation I told, one is the indexed, another is the sequential, another linked representation. Array for example is an indexed representation, linked list for example is a linked representation or sequential representation rather and tree structure is follows a linked representation. What is a linked representation? That we shall understand when we will discuss about the concept of tree that is there in the theory of computer science, data structure point of view.

(Refer Slide Time: 32:52)

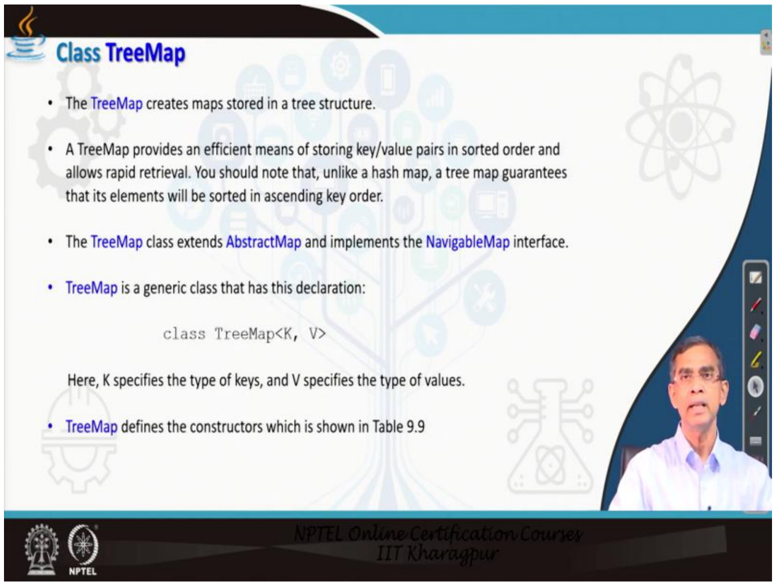


Map classes

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.

Anyway so tree map is basically the extension of the abstract class, it basically follows a tree concept, a tree structure to store the key value pairs for the different objects.

(Refer Slide Time: 33:07)



Class TreeMap

- The TreeMap creates maps stored in a tree structure.
- A TreeMap provides an efficient means of storing key/value pairs in sorted order and allows rapid retrieval. You should note that, unlike a hash map, a tree map guarantees that its elements will be sorted in ascending key order.
- The TreeMap class extends AbstractMap and implements the NavigableMap interface.
- TreeMap is a generic class that has this declaration:

```
class TreeMap<K, V>
```

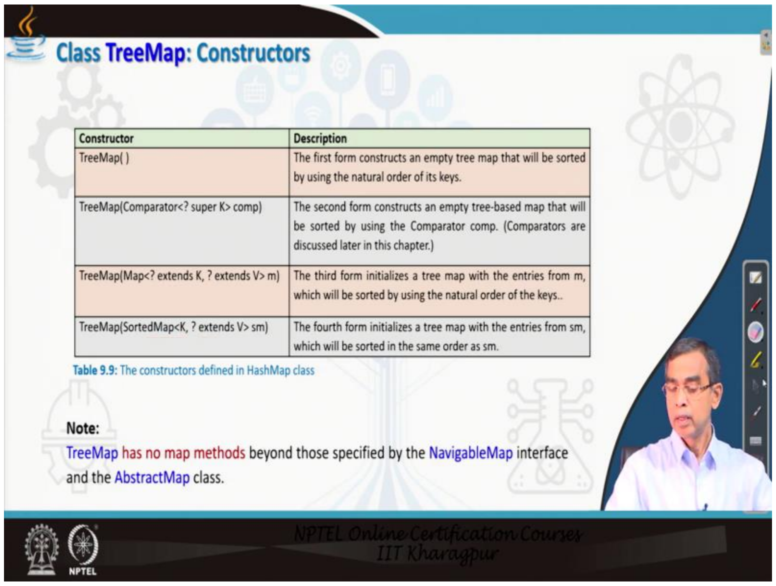
Here, K specifies the type of keys, and V specifies the type of values.

- TreeMap defines the constructors which is shown in Table 9.9

NPTEL Online Certification Courses
IIT Kharagpur

So, two elements are to be there, so it is basically a collection by virtue of key objects and value objects for each key value pairs of objects.

(Refer Slide Time: 33:18)



Class TreeMap: Constructors

Constructor	Description
TreeMap( )	The first form constructs an empty tree map that will be sorted by using the natural order of its keys.
TreeMap(Comparator<? super K> comp)	The second form constructs an empty tree-based map that will be sorted by using the Comparator comp. (Comparators are discussed later in this chapter.)
TreeMap(Map<? extends K, ? extends V> m)	The third form initializes a tree map with the entries from m, which will be sorted by using the natural order of the keys..
TreeMap(SortedMap<K, ? extends V> sm)	The fourth form initializes a tree map with the entries from sm, which will be sorted in the same order as sm.

Table 9.9: The constructors defined in HashMap class

Note:
TreeMap has no map methods beyond those specified by the NavigableMap interface and the AbstractMap class.

NPTEL Online Certification Courses
IIT Kharagpur

Now it also includes several constructors. The constructor, first one is the default constructor tree map, the second is basically another constructor by which how the tree can be built, there actually you have to define the comparator method because if you want to store, you just define objects which has not the defined method of comparator.

Then you should define this comparator method, in that you just define type and then use it here. And then tree map, it is basically the last constructor you can note. This basically tree map, sorted map K extends Vm. So here also the tree can be store, tree collection can be stored using the concept of sorted map that we have already discussed about the interface.

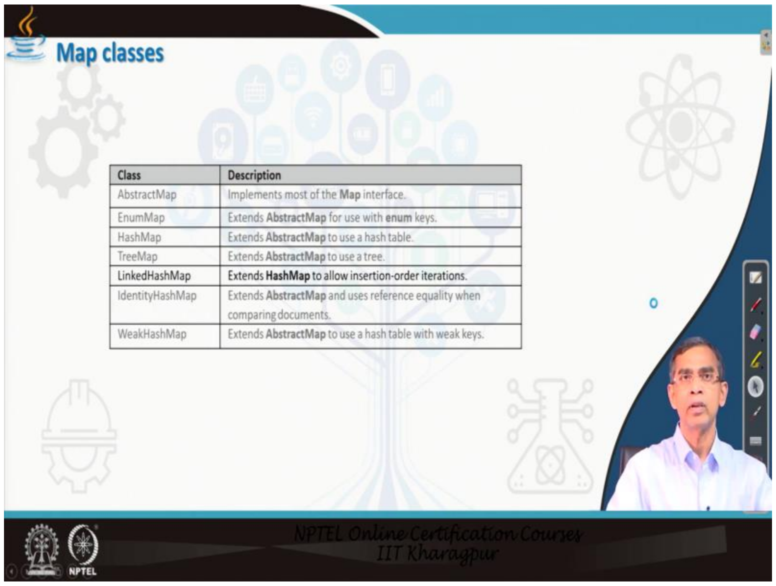
So that can be easier and again the tree map methods, if you see there are not any specific method but those are the method specified by the navigable interface are there. So this means that tree map methods includes all the methods those are there in the map interface as well as those are there in navigable map interface is the part of this interface. So this is the tree map, now next is basically linked hashmap class.

(Refer Slide Time: 34:47)

LinkedHashMap Class

Linked hashmap pass use the linked representation, it is actually a sequential representation I can say, sequential representation of the map collection object.

(Refer Slide Time: 34:57)



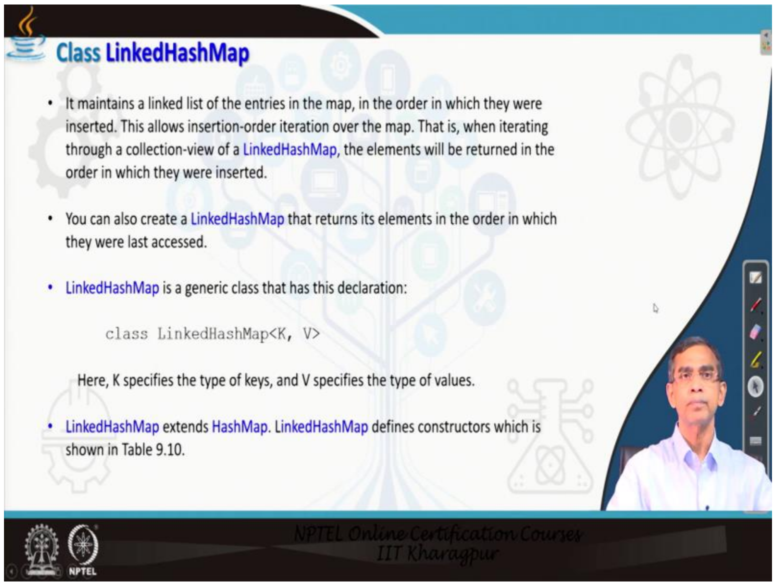
Map classes

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.

NPTEL Online Certification Courses
IIT Kharagpur

So here idea is that, it is basically allow insert, insertion order iteration. Actually the idea is that if you see the sequential representation, if you want to add a new entry into a collection, it always add in a particular order. So ordering is basically in what order the element will be inserted. So here all the entries will be stored in an order but that order is not according to the key values. This order is according to the order of insertion. So the first element which is inserted should be at the head or the front and then the last element will be at the tail, at the end actually, so it is layer. And in, so this way the elements are maintained in a sorted order.

(Refer Slide Time: 35:53)



Class LinkedHashMap

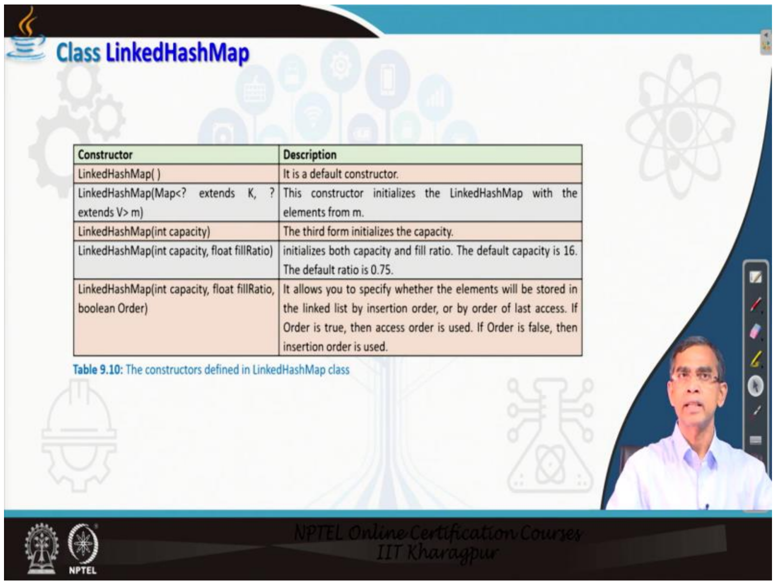
- It maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map. That is, when iterating through a collection-view of a LinkedHashMap, the elements will be returned in the order in which they were inserted.
- You can also create a LinkedHashMap that returns its elements in the order in which they were last accessed.
- LinkedHashMap is a generic class that has this declaration:

```
class LinkedHashMap<K, V>
```

Here, K specifies the type of keys, and V specifies the type of values.

- LinkedHashMap extends HashMap. LinkedHashMap defines constructors which is shown in Table 9.10.

NPTEL Online Certification Courses
IIT Kharagpur



Class LinkedHashMap

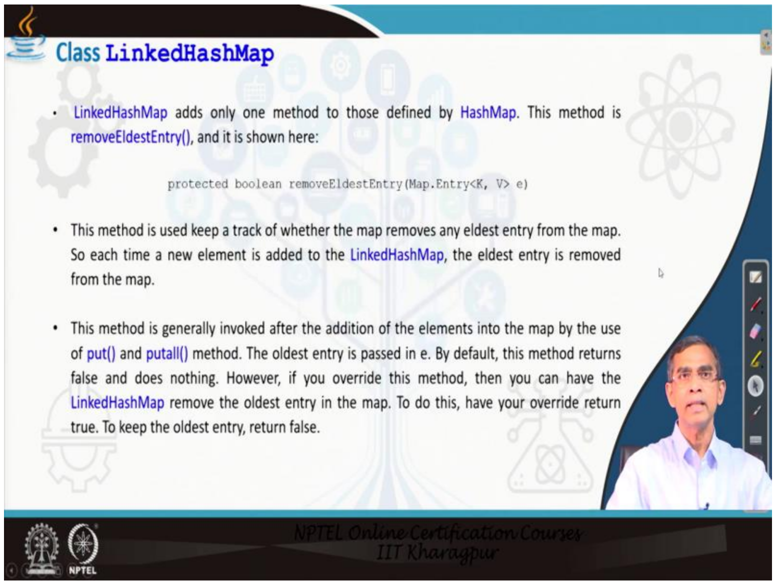
Constructor	Description
LinkedHashMap( )	It is a default constructor.
LinkedHashMap(Map<? extends K, ? extends V> m)	This constructor initializes the LinkedHashMap with the elements from m.
LinkedHashMap(int capacity)	The third form initializes the capacity.
LinkedHashMap(int capacity, float fillRatio)	initializes both capacity and fill ratio. The default capacity is 16. The default ratio is 0.75.
LinkedHashMap(int capacity, float fillRatio, boolean Order)	It allows you to specify whether the elements will be stored in the linked list by insertion order, or by order of last access. If Order is true, then access order is used. If Order is false, then insertion order is used.

Table 9.10: The constructors defined in LinkedHashMap class

NPTEL Online Certification Courses
IIT Kharagpur

There are few constructors belongs to this class. The constructors are defined here, the default, first one is the default constructors. The second one is basically the constructor suppling key values and object value, both together if you want to add more entities. Then the next, the third constructor is basically creating a map collection specifying the size, default size is 16. And then the last one is basically size and fill ratio both can be mentioned in order to create a new collection of type linked hashmap.

(Refer Slide Time: 36:34)



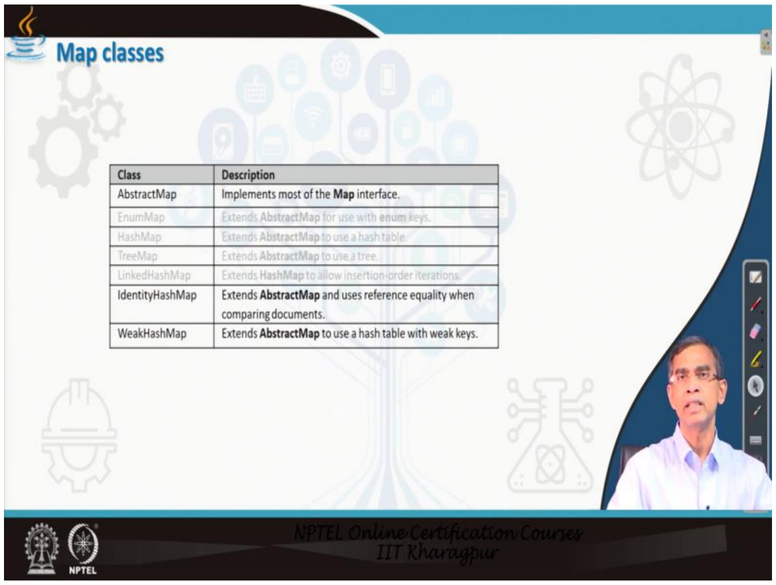
Class LinkedHashMap

- LinkedHashMap adds only one method to those defined by HashMap. This method is removeEldestEntry(), and it is shown here:

```
protected boolean removeEldestEntry(Map.Entry<K, V> e)
```

- This method is used keep a track of whether the map removes any eldest entry from the map. So each time a new element is added to the LinkedHashMap, the eldest entry is removed from the map.
- This method is generally invoked after the addition of the elements into the map by the use of put() and putall() method. The oldest entry is passed in e. By default, this method returns false and does nothing. However, if you override this method, then you can have the LinkedHashMap remove the oldest entry in the map. To do this, have your override return true. To keep the oldest entry, return false.

NPTEL Online Certification Courses
IIT Kharagpur

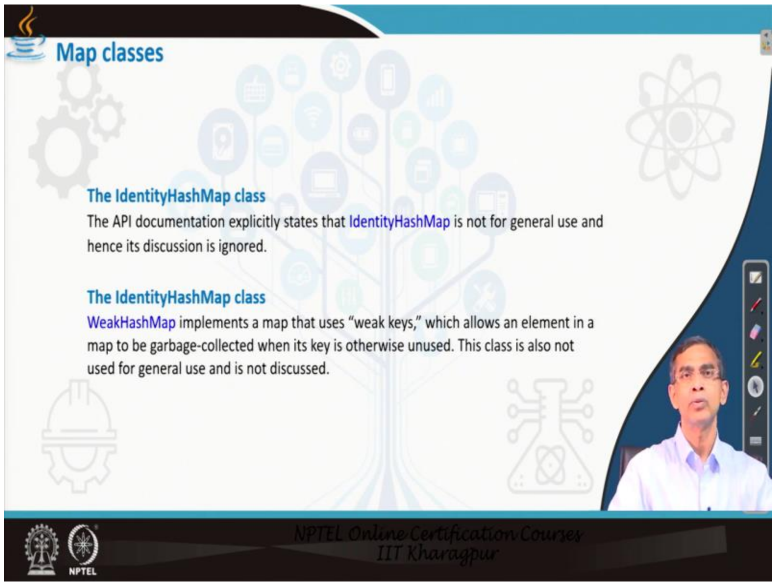


Map classes

Class	Description
AbstractMap	Implements most of the **Map** interface.
EnumMap	Extends **AbstractMap** for use with **enum** keys.
HashMap	Extends **AbstractMap** to use a hash table.
TreeMap	Extends **AbstractMap** to use a tree.
LinkedHashMap	Extends **HashMap** to allow insertion-order iterations.
IdentityHashMap	Extends **AbstractMap** and uses reference equality when comparing documents.
WeakHashMap	Extends **AbstractMap** to use a hash table with weak keys.

Now there are certain classes also there. All the classes those are there in the map method are basically defined for this linked hashmap method. It does not have any special methods of it there.

(Refer Slide Time: 36:50)



Map classes

The IdentityHashMap class

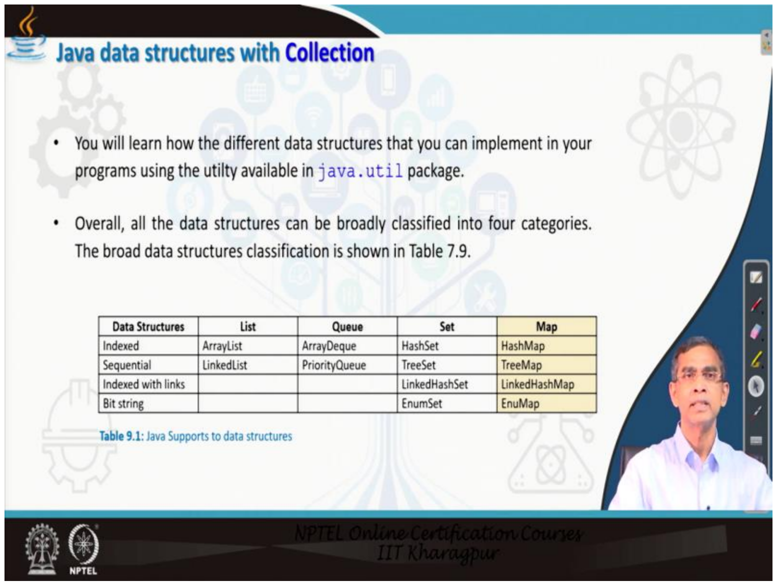
The API documentation explicitly states that IdentityHashMap is not for general use and hence its discussion is ignored.

The IdentityHashMap class

WeakHashMap implements a map that uses "weak keys," which allows an element in a map to be garbage-collected when its key is otherwise unused. This class is also not used for general use and is not discussed.

And two other methods, two other classes like identity hashmap class and, sorry the first one is weak hashmap class and the next one is identity hashmap class. These two classes are basically for the use of Java developer not for the programmer. So it is basically at the system level development, the Java developer defines classes to be used for maintaining the Java packages and other, so we should not, as the programmer does not have any interest of it, so we should not discuss in details about them.

(Refer Slide Time: 37:26)



Java data structures with Collection

- You will learn how the different data structures that you can implement in your programs using the utilty 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.

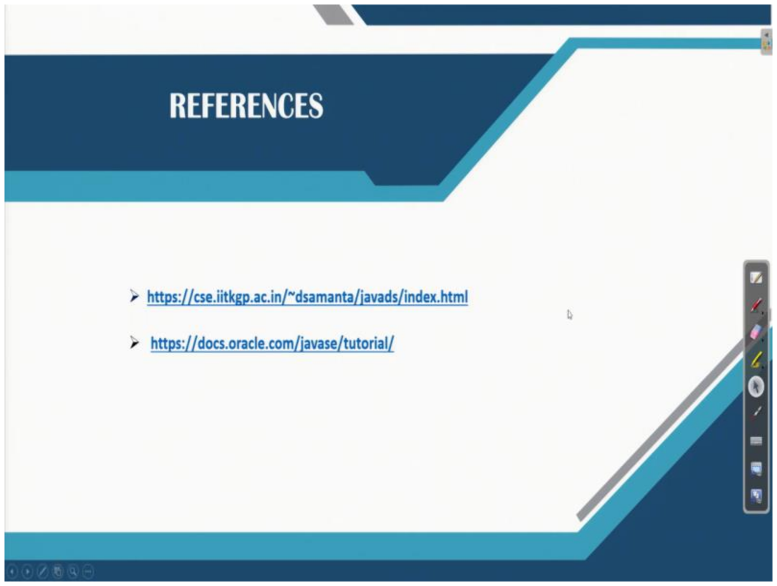
Data Structures	List	Queue	Set	Map
Indexed	ArrayList	ArrayDeque	HashSet	HashMap
Sequential	LinkedList	PriorityQueue	TreeSet	TreeMap
Indexed with links			LinkedHashSet	LinkedHashMap
Bit string			EnumSet	EnuMap

Table 9.1: Java Supports to data structures

NPTEL Online Certification Courses
IIT Kharagpur

Now finally I will come to here that whatever the map collections we have discussed, they broadly can be categorized into four, one is hashmap which is of type indexed data structure is follows, tree map is basically follow linked structure and then the linked hash structure is follow basically sequential concept and the enum map is basically follow a string structure or bit string form actually it is there.

(Refer Slide Time: 37:56)



REFERENCES

- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html
- https://docs.oracle.com/javase/tutorial/

Now regarding the discussion about all these collection concepts you can consult the link that is given, the first link is basically useful link one because it is discuss in a very user friendly, reader friendly manner. For advanced user the oracle tutorial document is very nice, you can go through it. In the oracle tutorial document also you can find the discussion along with very small piece of examples to elaborating many things are there. Third that there are many what is called the net is also available in the net like geeks for geeks, Java T point or all these things, there also very nice representation for easy to understand concept it is there. Thank you for your attention.