1. **What all collection you used in your project ?**



I worked on the List Interface, Set Interface, and Map Interface. I used the Implementation of List as ArrayList and LinkedList. As part of Set Implementation, I have worked on HashSet, LinkedHashSet and TreeSet. As part of Map Implementation, I worked on HashMap, LinkedHashMap and TreeMap.

Also, there is few more implementation class like CopyOnWriteArrayList, CopyOnWriteArraySet and ConcurrentHashMap. Which introduced in **java.util.concurrent.\*** package.

1. **What is the difference between List and Set ?**
2. List will allow us to store a duplicate object where Set will not allow to store a duplicate object.
3. List works on Indexed based where Set works based on the hash value.
4. Always we need to use List when you want to frequently access the elements the by using the index. Where Set is used when you want to design a collection of distinct elements.
5. So, the moral of using this list and set if u want to allow duplicate object, then go for list or if you want to maintain a unique object or unique set then u can go for set.
6. **What is the difference between ArrayList and LinkedList ?**

ArrayList and LinkedList is the 2 implementations of a List Interface.

1. ArrayList Internally uses a dynamic array to store the elements where LinkedList Internally uses a doubly linked list to store the elements.
2. Manipulation with ArrayList is slow because it internally uses an array. If any elements is removed from an array all the bits are shifted in memory. Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3. ArrayList is better for storing and accessing data. LinkedList is better for manipulating data.
4. **List object creation scenario**

**ArrayList arrayList = new ArrayList<String>();**

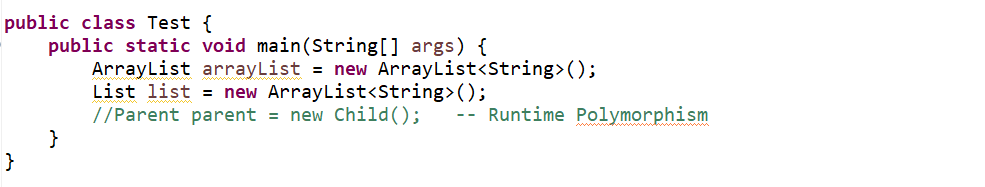
**List<String> list = new ArrayList<>();**

What is the difference between these 2 and which one is preferrable to use a list implementation object.

In case of first scenario here we are directly using an array list implementation which is tightly coupled but in case of second syntax you can say List is an interface with a reference of list or with a reference of parent we are creating a object of its implementation class( new ArrayList()).

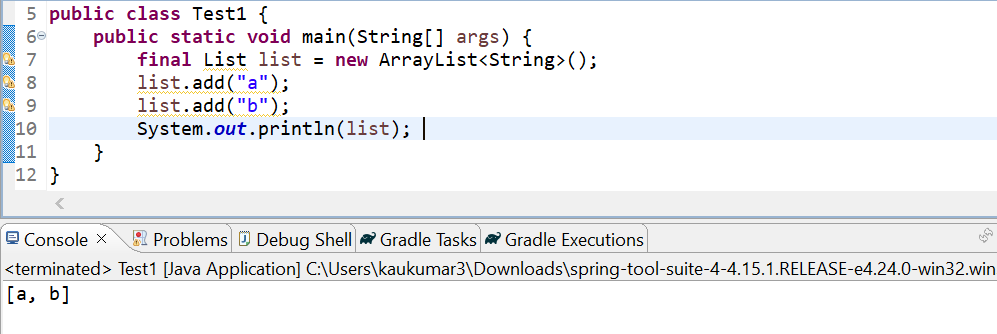
In future if I want to create the LinkedList Object then we can simply write new LinkedList() instead of ArrayList. So, no need to change the complete signature. That’s what we understand as part of Runtime Polymorphism. With the help of parent reference, we are creating the object of child. That is what ArrayList is a child and List is an interface.

1. **Can’t we use List instead of List<String> ?**

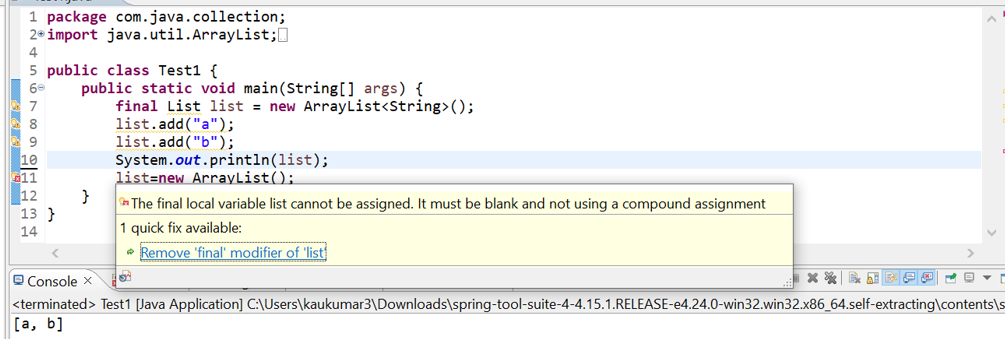


Its always recommended to use the Type Generic that’s why my List will only allow to store the String Object. So, there will be no Type Cast issue we will face in future if you maintain the Generic.

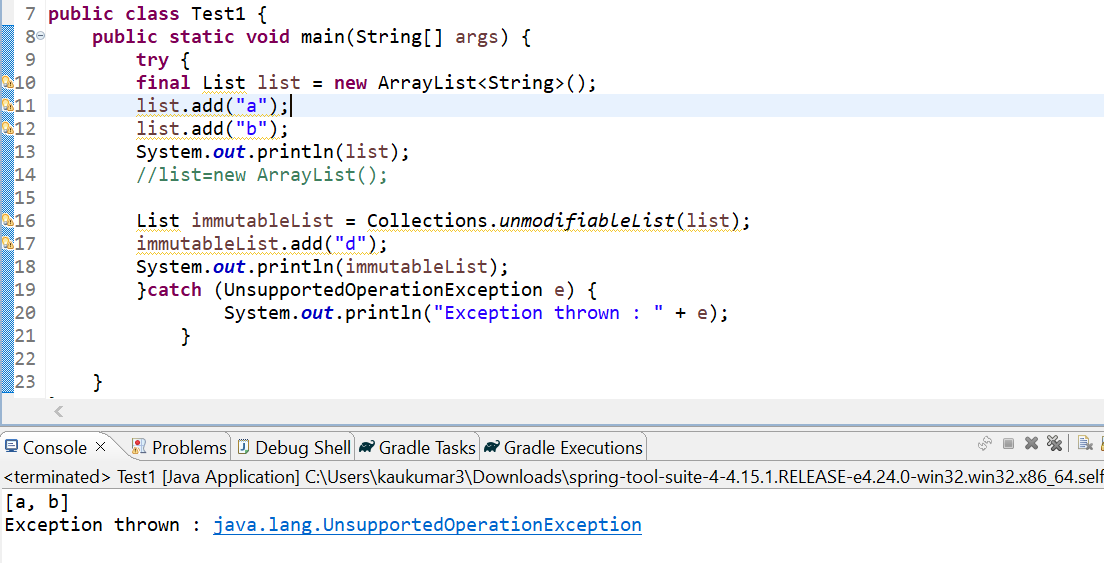
1. **Declaring a List field with the final keyword ?**



Here I declared the list as a final now interviewer will try to understand after declaring List as a final whether you are able to add the object to this list or not. Even though we declared it as a final we can modified it there is no immutability. Still, we can modify but we can’t re-assign this list. With the same reference we can’t create another object. But final doesn’t mean we can’t modify it.



If you don’t want to modify the List, then you can use Collections.*unmodifiableList*(list)

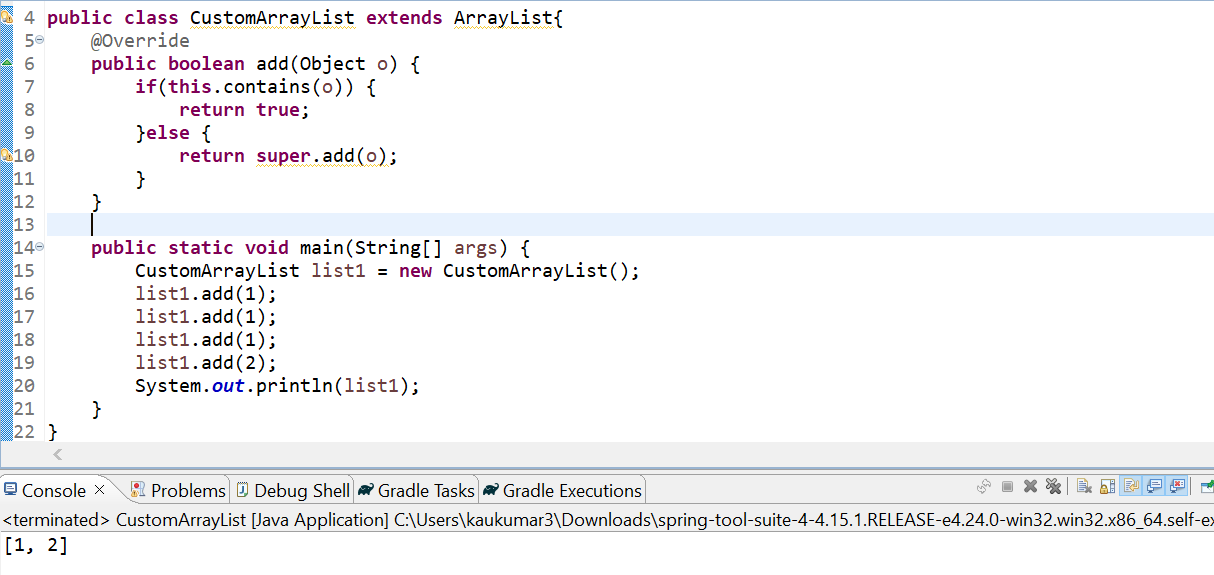


1. **How Can I write Custom ArrayList where I don’t want to allow duplicate ?**

We know array list will allow duplicate. But I want to create my own ArrayList where I stop allowing the duplicates.

I have created a class called **CustomArrayList** we just need to extend it from **ArrayList**. Then we will use its own method from the ArrayList method, and we will override the method and we will provide our own logic. So, I will just override the add() method from the array list.

Now let’s add couple of duplicate values in our customize array list.

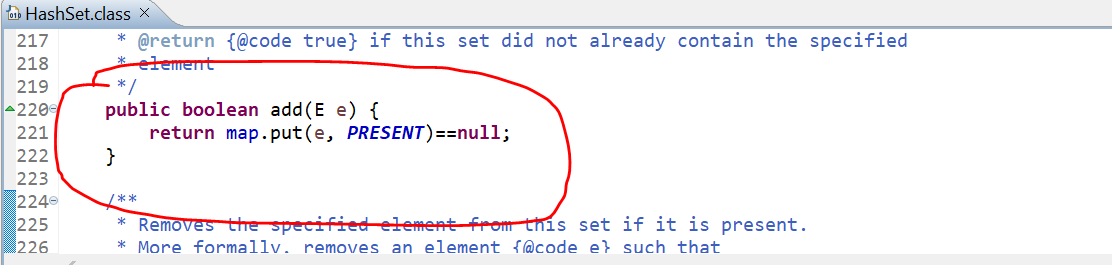


So, we can see it doesn’t allow duplicate objects in our list.

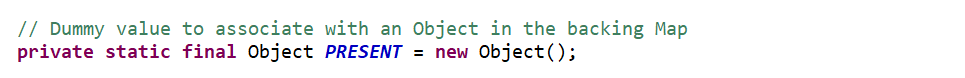
So, this is how you can create your own custom array list by extending it from ArrayList or any LinkedList and you can provide your own implementation to stop allowing duplicates.

1. **Why Set doesn’t allow duplicate Element ?**

As we know ArrayList will allow duplicate elements, but we can customize, and we can maintain a non-duplicate object in Array List by creating our own class. But why set doesn’t allow duplicate let’s see the **HashSet** implementation…



If you observed the object of add method of HashSet. It will add as a key of a Map. Which means Set method internally uses a Map. So, add method of HashSet internally uses the map object to store the value. where we considered the passed argument as a key and Value as a PRESENT which is nothing but a dummy object value.

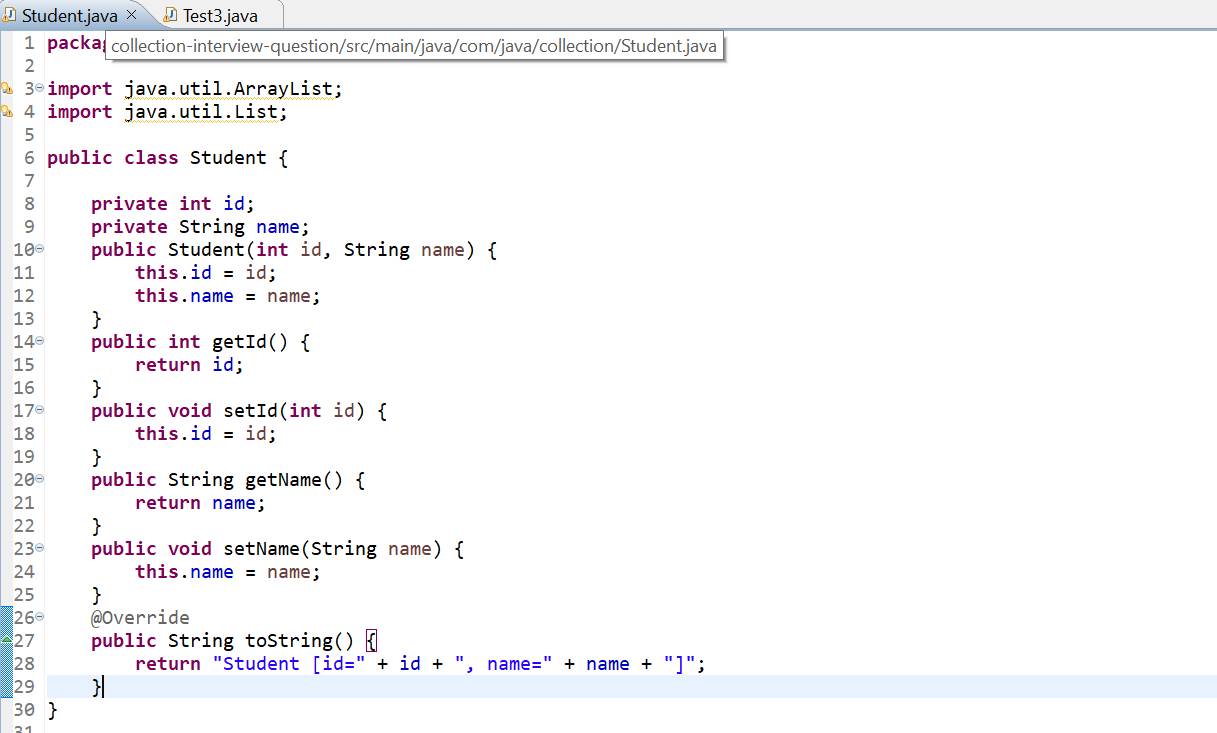


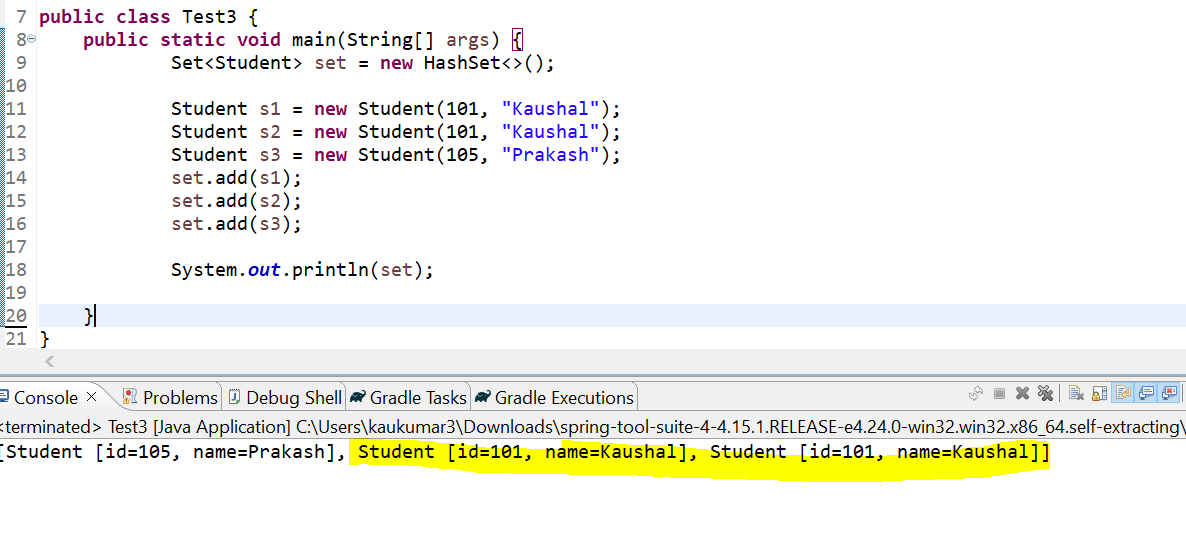
So **PRESENT** is a dummy object. It means Set implementation internally uses a Map to store object as a key, as we know map key will not allow a duplicate element.

1. **Does Set implementation always follow the same rule it does not allow the duplicate elements ?**

No, there is some certain rule if that is not followed by any of the HashSet or any of the Set implementation or Map implementation then it will allow duplicate object.

Now let me explain that scenario. Let’s assume I have a Student class.



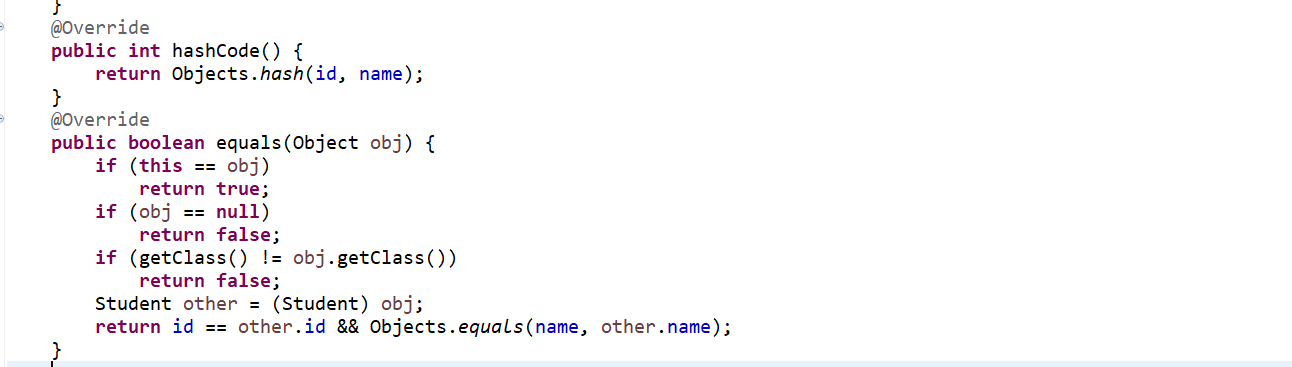


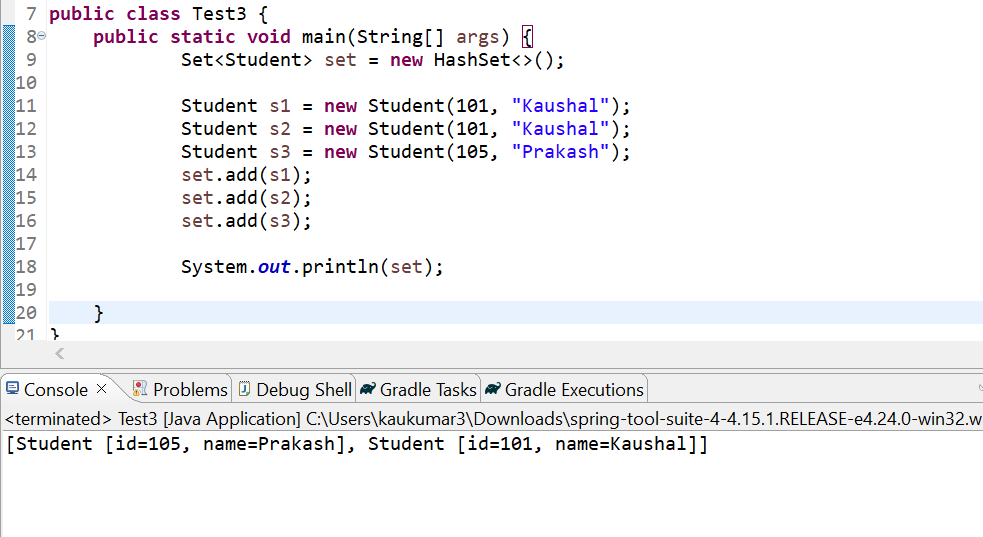
As per rule this Set should not allow duplicate object, right? Bcz s1 and s2 are the same object it should print only one of them and s3. But we can see all the 3 objects.

Which means Set is not following the rule it allowing duplicates. So, there is a difference.

If you are using Set with a primitive data type, then its fine no need to override **equals()** and **hashcode**() methods. But if you are using any Custom object or Wrapper Class then you must need to override **equals**() and **hashCode**() methods.

So, let’s override equals() and hashCode() methods.

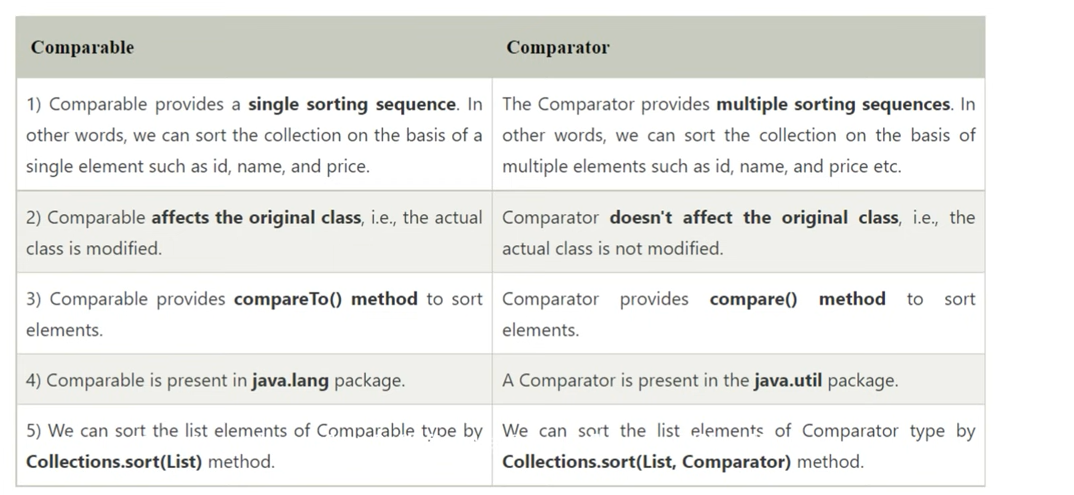




Now we are getting 2 objects. Which means even though you are using HashSet or HashMap that’s not guarantee it will not allow duplicate but yes if your custom object is overriding equals() and hashCode() methods. If they follow the contract of **equals**() and **hashCode**() then always you will get the unique object. There will be no duplicate.

**So, This is one of the Interview Question To explain the contract between equals(0 and hashCode() methods…**

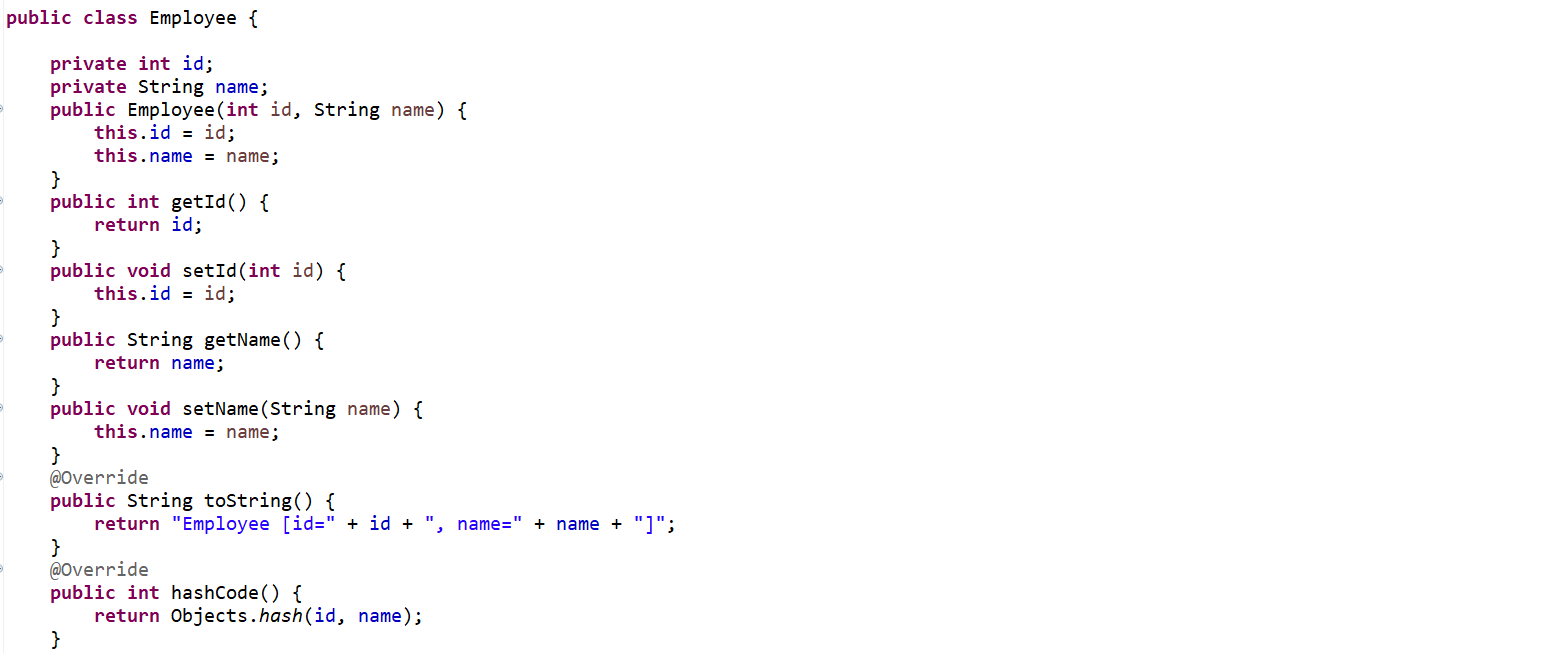
1. **What is the difference between Comparable and Comparator ?**



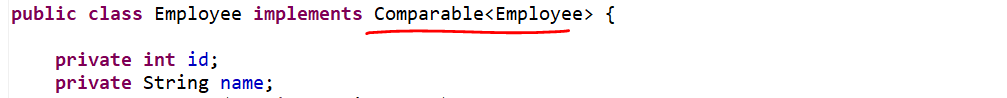
If we want to sort the collection based on a single element or based on only single fields, we should go for Comparable but if we want to sort based on multiple parameters with the help of Comparable, we can’t achieve it we need to go for Comparator.

Comparable affects the original class bcz we are implementing Comparable and overriding the Comparable method but if we are using Comparator, we were using a separate class.

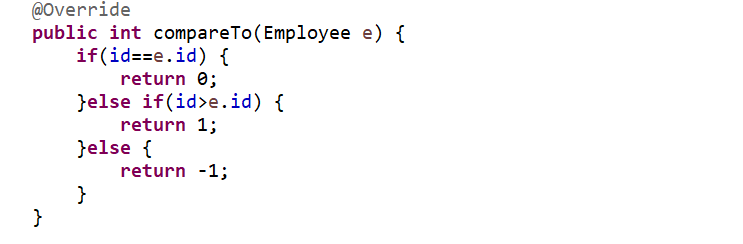
Let’s create an Employee Class I want to sort based on Id.



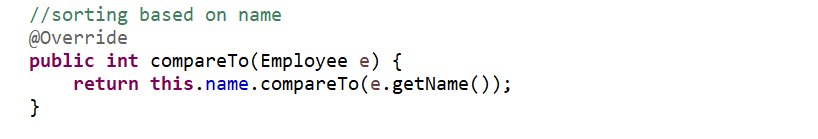
If I will use **Comparable,** then it will directly affect my actual class which is **Employee.**



**Sorting based on Id(integer)**



**Sorting Based on name(String)**

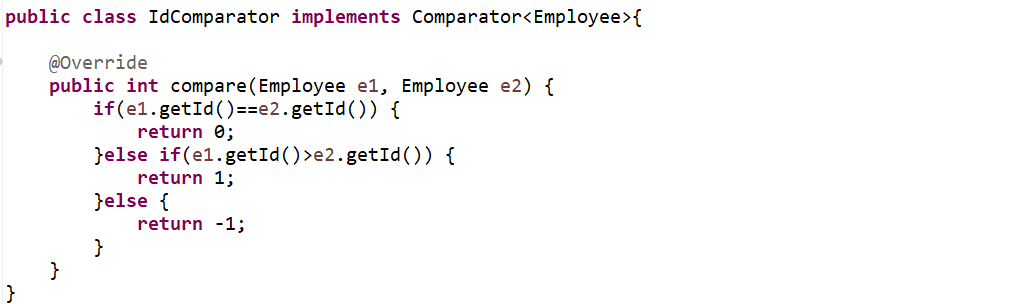


Now in future I got a requirement just change the sorting mechanism based on some other field now again I need to change this code so in case of comparable its not dynamic its always like hard coded.

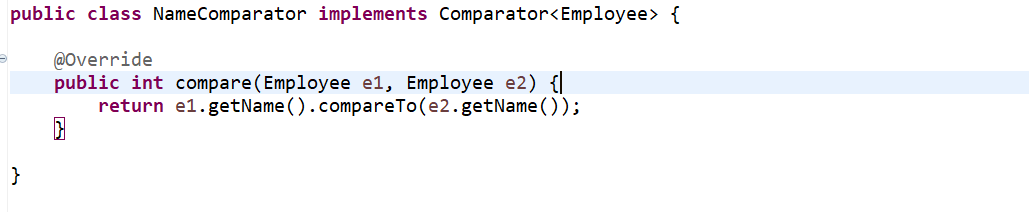
This will recommend if u want to do based on single sorting mechanism.

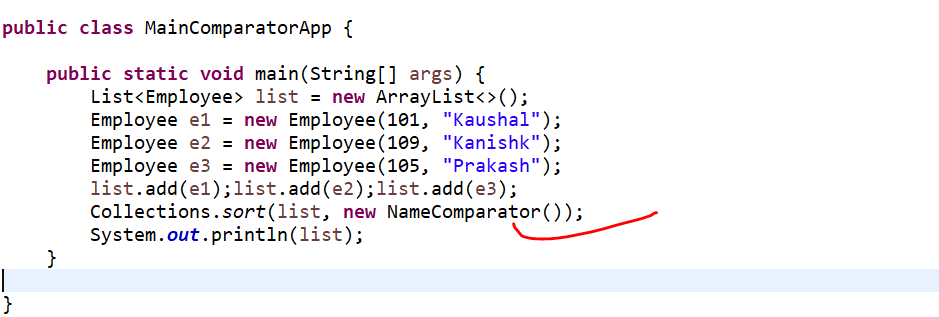
If I want to sort based on multiple parameters, we should go for Comparator.

//sort based on Id

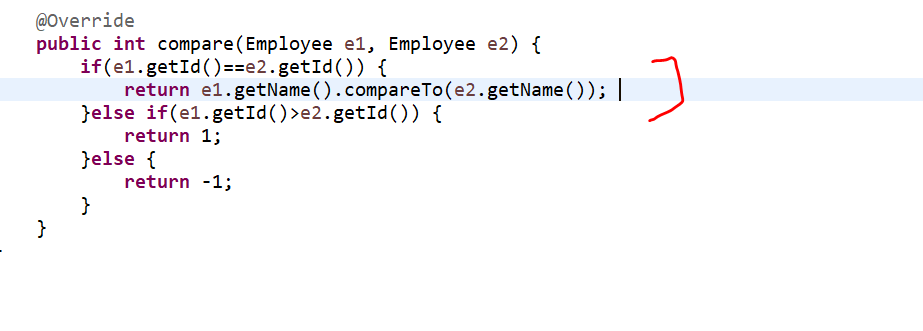


//sort based on name

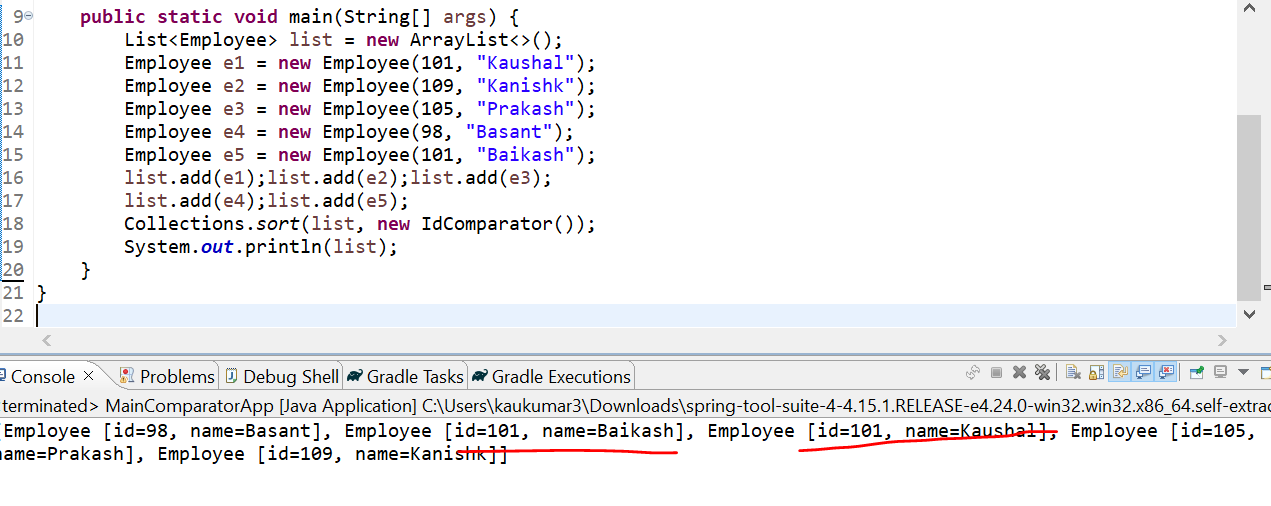




1. **I just want to sort based on Id if I found Id is same then I will go with the name sorting ?**



So, this comparator basically sort based on Id if it found both the id of its object is same then it will sort based on the name.



1. **Multi Comparing using Comparator Scenario ?**

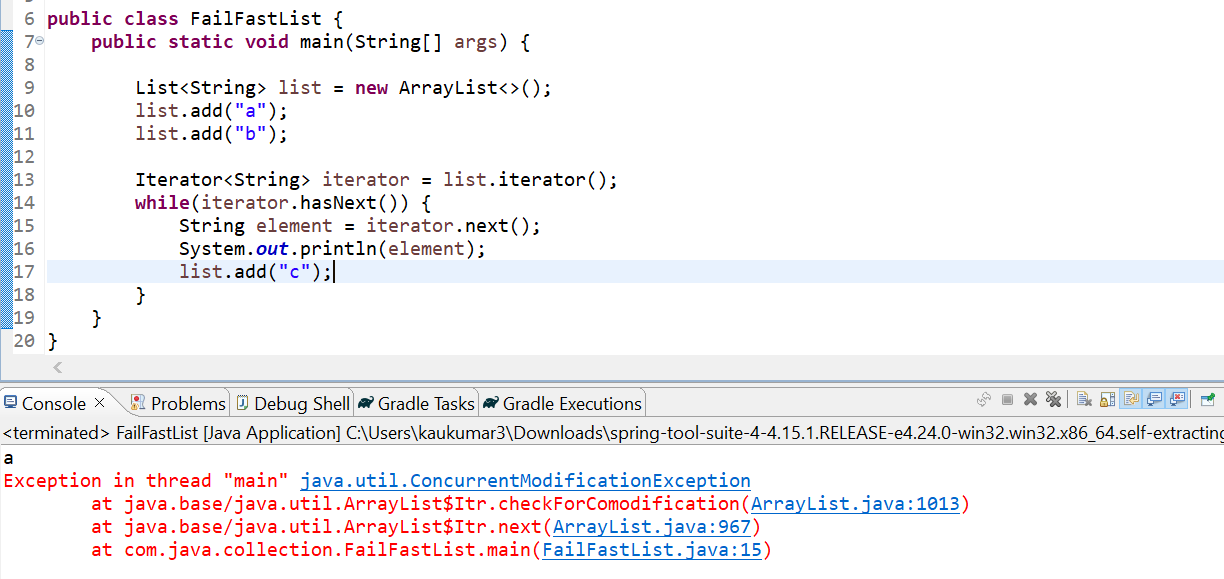
If we sort based on Id if we found Id is same, then sort with name.

1. **What is difference between fail-fast and fail-safe Iterator ?**

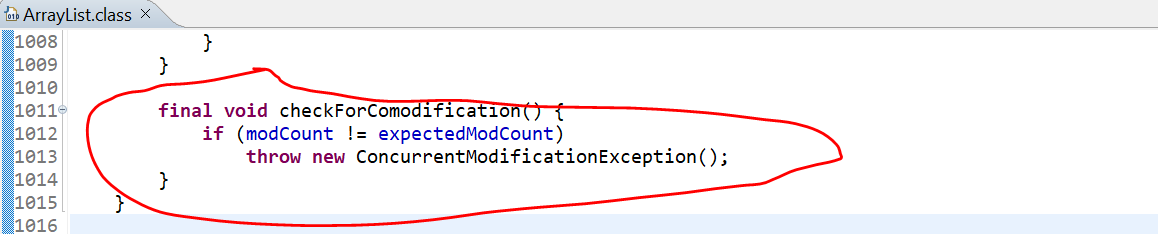
A Iterator which will fail fast when we do any modification while iterating a collection is called fail-fast iterator. Ex – **ArrayList**, **HashMap** and **Vector**.

Let’s say I have one collection and I was trying to iterating it. While iterating in middle I am trying to modify something on a same collection object then we will get some exception which is **ConcurrentModificationException.** That’s the reason it Is called fail fast iterator.

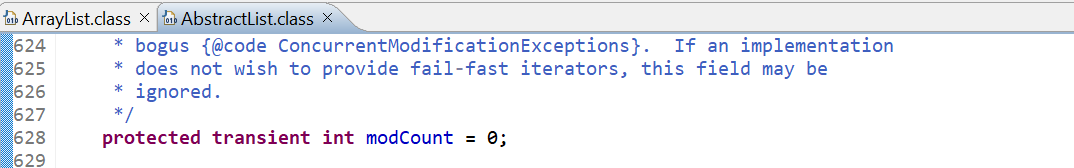
A Iterator who allow us to modify in middle while iterating a collection is called fail-safe iterator. Ex- **CopyOnWriteArrayList**, **CopyOnWriteArraySet** and **ConcurrentHashMap**. These all are introduced in **java.util.concurrent** package.



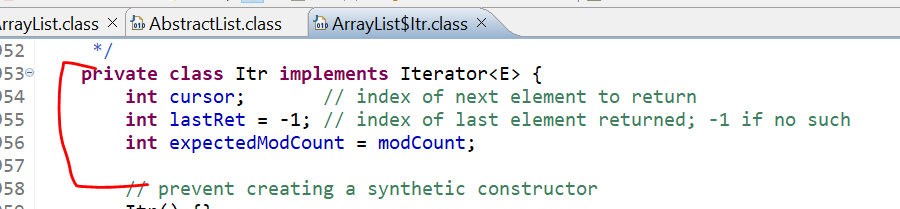
Now one thread is iterating and I am trying to add something in middle I am getting **ConcurrentModificationException** the reason here internally this iterator use a **modCount** so any changes on the collection will be reflected in the **modCount** it will change. At the end it will evaluate the **modCount** if there is a change found in a **modCount**. Initially just assume the **modCount** size is 0. And here I was adding something then the **modCount** got changed to 1. If there is any difference found in **modCount** then immediately it will throw the **ConcurrentModificationException**. So to prove it let me go to this implementation of ArrayList.



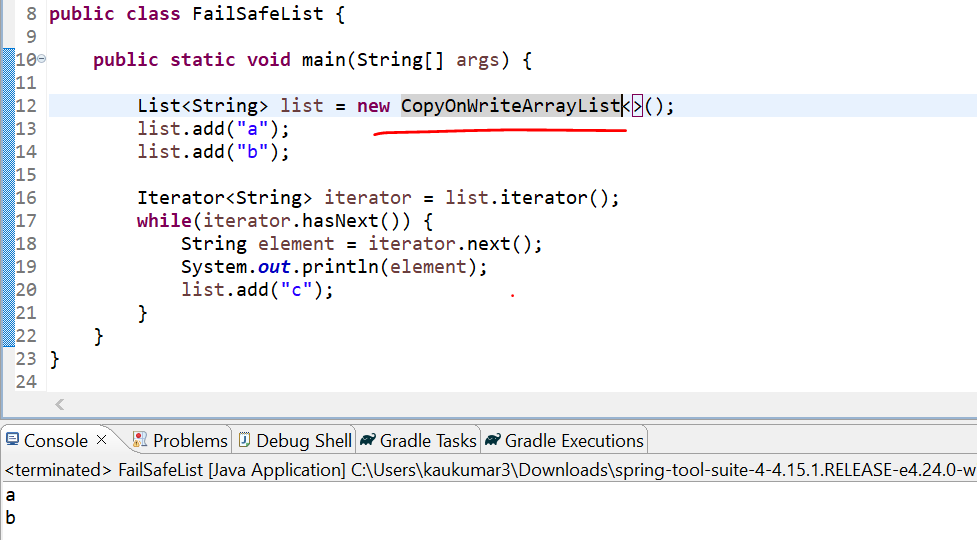
If u go to **modCount** initially it assigned to 0.



If u change anything in your collection, then immediately it will increment by 1.



How can we avoid this, so to avoid just use **CopyOnWriteArrayList**.



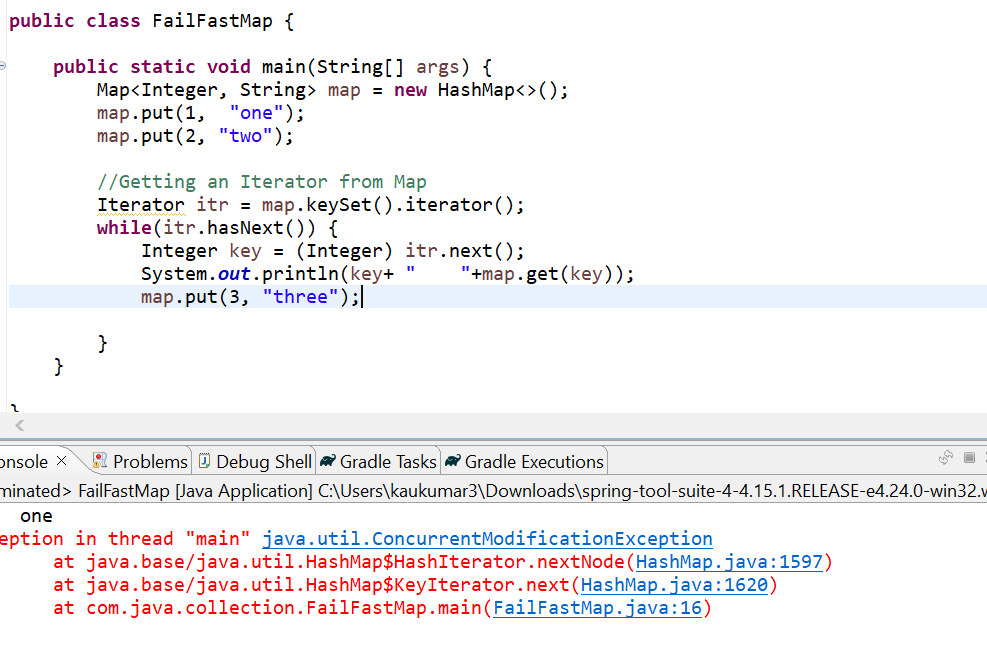
So, while iterating we are able to modify the collection that is why its called fail-safe or non-fail fast iterator.

So, now the question is how it work on **CopyOnWriteArrayList** either its going to check **modCount** or not.

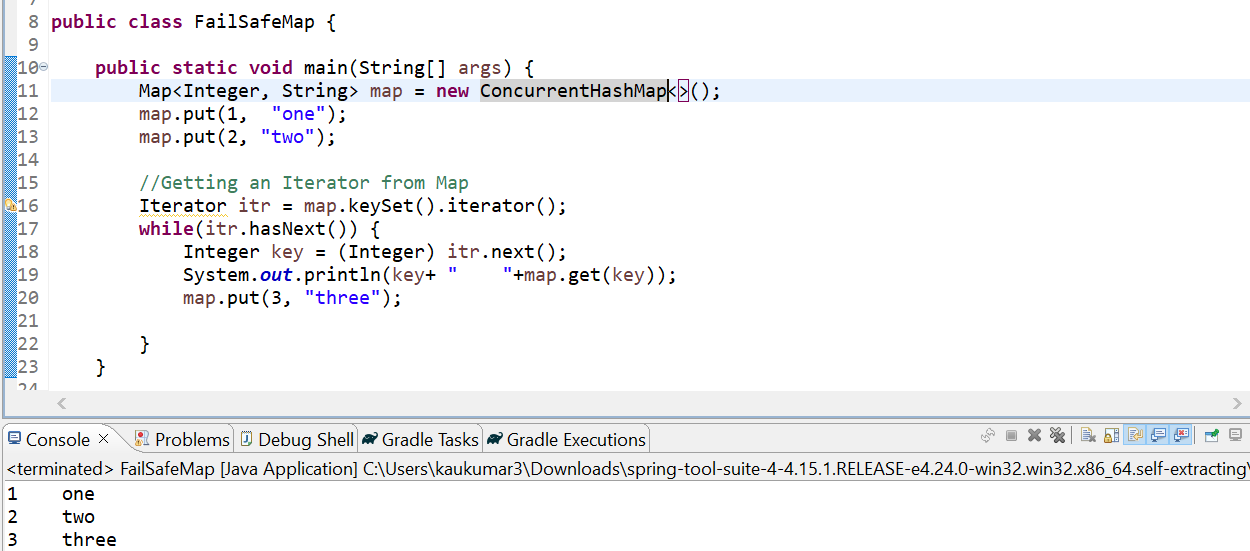
Yes it is going to check the mod count but if there is any difference in **modCount** immediately it will create a cloned copy of collection and any update will happened to that cloned copy object instead of this actual collection object.

So, to prove that if u observed here, I used CopyOnArrayList and I add **list.add(“c”).** but it will not print because there will be 2 iterations as we have 2 elements a and b so next while iterating only, we are adding one more element c. so, there will be a next iteration but in output only we can see 2 iterations.

Which means its not adding to this list object. It will add to the **copy of list** object.so, internally it will use the copy of list object when there will be any difference in mod count. Similarly, if I will go to the FailFastMap.



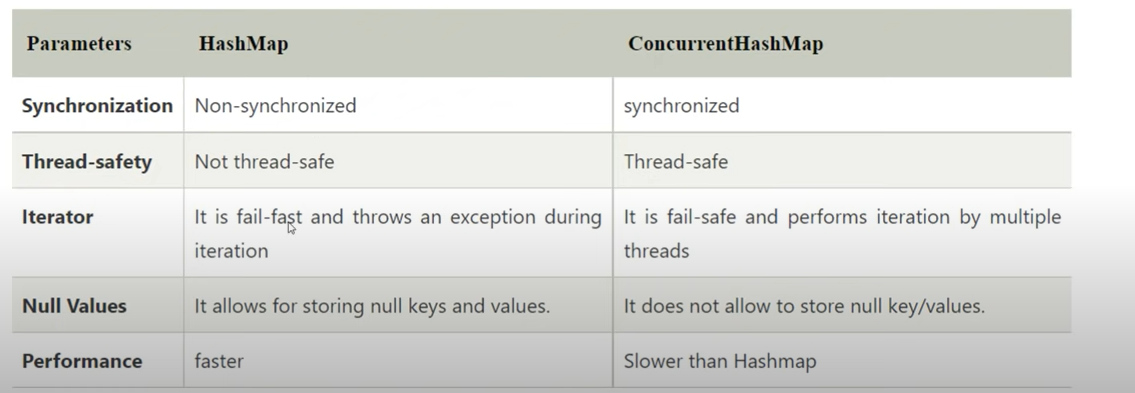
We are getting exception now what we can do instead of **HashMap** let us use **ConcurrentHashMap**.



But in case of ConcurrentHashMap We are not getting error here bcz it is not using your cloned copy of Collection implementation. Since we are using Map here so there will be no copy if u can observe we are getting all the 3 objects. But in case of CopyOnWriteArrayList we are not getting all values bcz it returns cloned copy of collection bcz when **modCount** is getter than **expectedModCount** it will create a cloned copy of collection instead of throwing **ConcurrentModificationException**. But in case of map the implementation is different there is no cloned copy concepts.

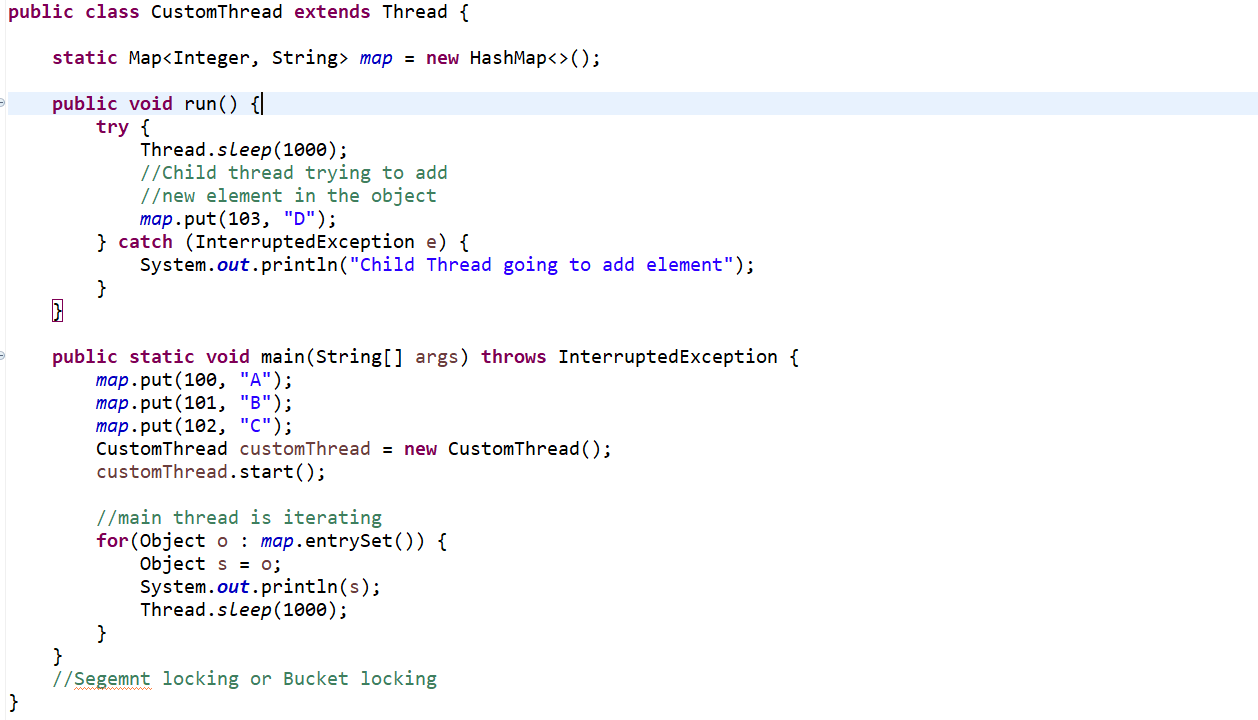
So, you can use ConcurrentHashMap whenever you want to access through parallel threads or while iterating you want to modify something you can go for **ConcurrentHashMap**.

1. **What is the need of ConcurrentHashMap and How it is different from HashMap ?**

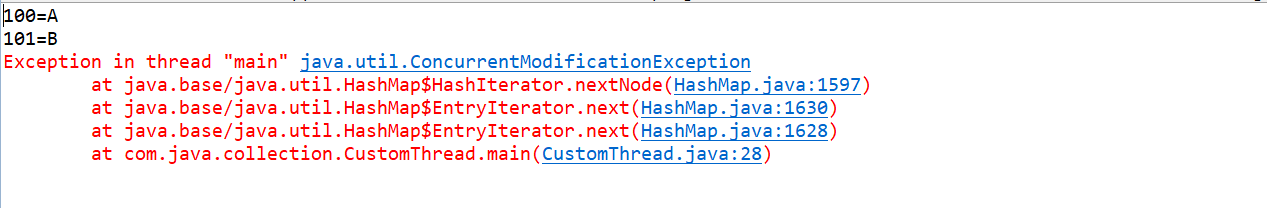


Let’s see one example. I am going to create a class called **CustomThread**.

Globally I declared one Map object. There is a main thread and another which I extends from Thread is called a Child thread. Inside child thread I am trying something to add in our map. Inside main thread I have added 100, 101 and 102. Then I created thread object. From main thread I am trying to iterate it. But another thread is trying to insert something.



If I run this code, I will get a **ConcurrentModificationException**.



So, if u observe I will get a **ConcurrentModificationException** bcz main thread is iterating the map object and another thread child thread adding something into the same map object at same time. That’s what we are getting **ConcurrentModificationException** because we are using HashMap.

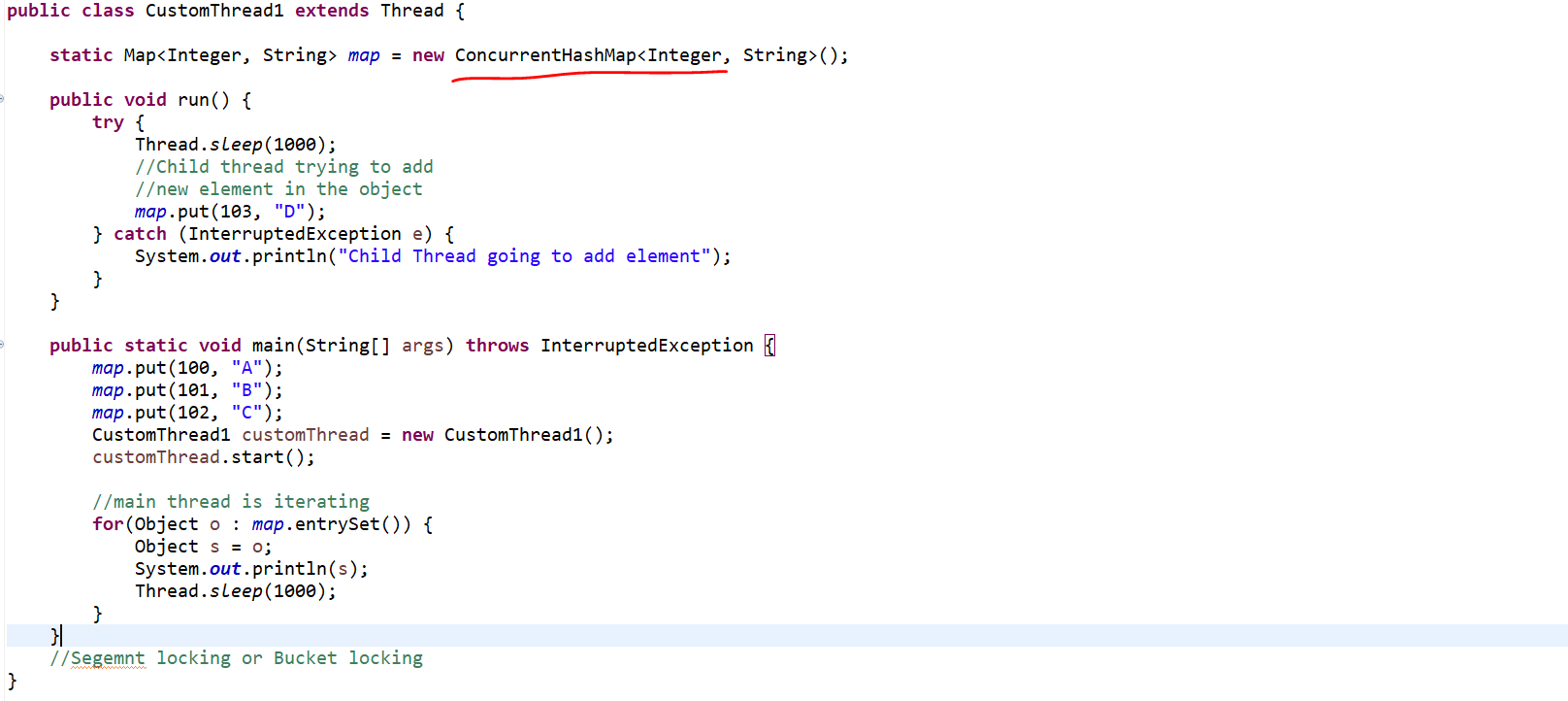
What happened internally this map apply lock on entire object now this map object is locked so that’s the reason one thread is iterating it got locked and it will not allow other thread to do something. That’s the reason we are getting **ConcurrentModificationException** in case of Map implementation HashMap.

Now instead of HashMap let’s use **ConcurrentHashMap**.

So, there are no exception, and we are getting all the output.

So, whenever you are using **ConcurrentHashMap** it allows multiple threads to access simultaneously. They can able to modify my underlying collection data structure. There will be no exception.

The reason here in case of **ConcurrentHashMap** i ConcurrentHashMap doesn’t allow lock on to the entire map object it apply lock on the **Segment** level.

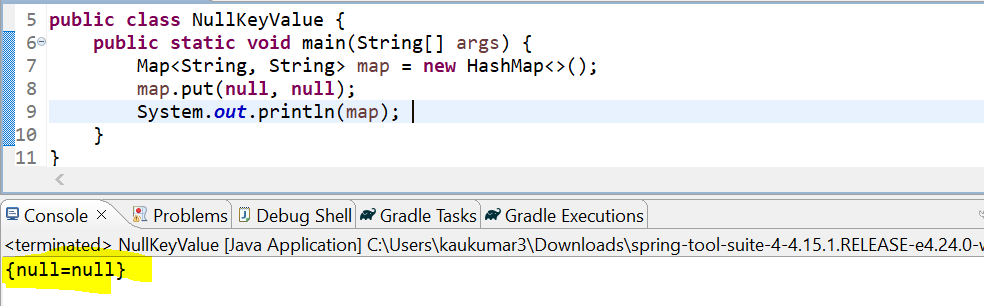


So, let’s say you are iterating it now I iterate the first entry since it starts from 0.so its try iterating 100 and A. in middle thread context switching happened so immediately it will go there and it will add another object. because we don’t know which thread will execute first parent or child thread. That will be depends on your thread scheduler. Then child thread will release the lock and main thread will add another entry and what **ConcurrentHashMap** does**.**

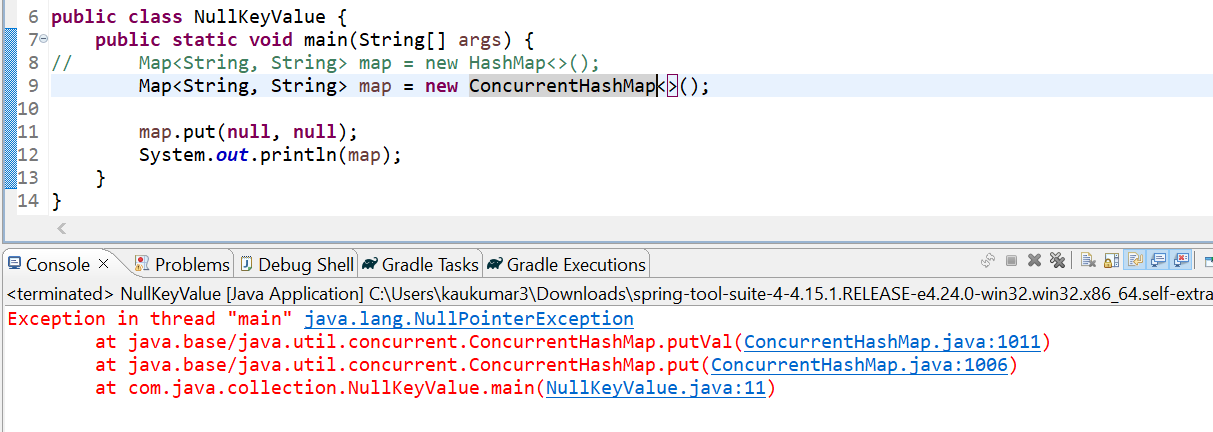
It just applies lock on the **Segment** Level. Rather than applying lock on the entire underlying collection object. So, HashMap is non synchronized but **ConcurrentHashMap** is synchronized.

HashMap apply lock on entire object where **ConcurrentHashMap** applies lock based on the segment. So that is called segment locking or bucket locking in **ConcurrentHashMap.**

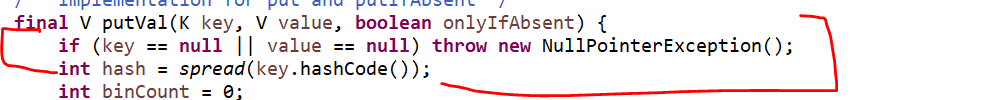
1. **HashMap allows null key or value but ConcurrentHashMap doesn’t allow?**



So, in case of key and value is null HashMap not giving any exception. Now if I will change it to the **ConcurrentHashMap** I will get the **NullPointerException**.

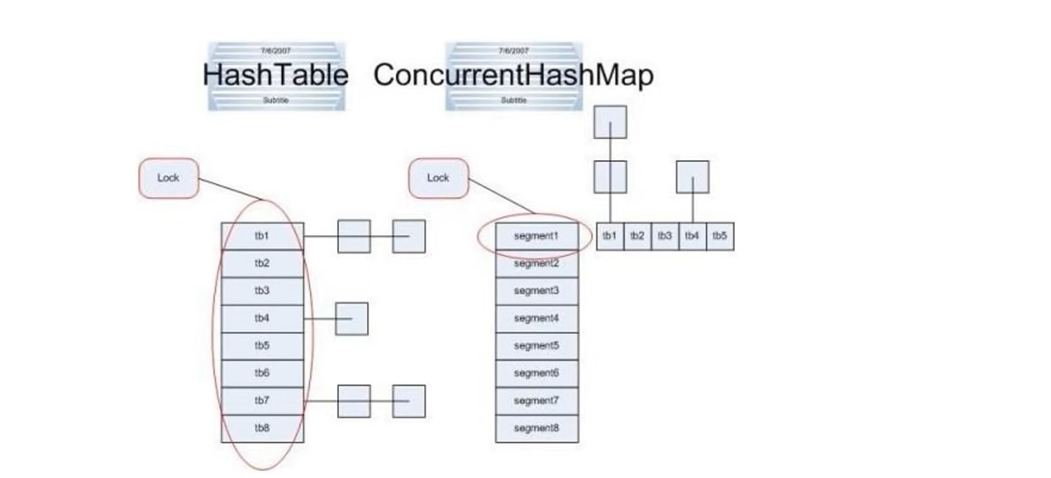


Now if I will go to this implementation



But this is not there in HashMap.

1. **If we have Hashtable which is already synchronized, then Why we need ConcurrentHashMap?**



Because Hashtable is already synchronized and ConcurrentHashMap is also synchronized then what is the difference between these 2. Even though Hashtable is synchronized but Locking mechanism still same as per HashMap it will lock whole underlying data structure. there will be no segment locking or bucket locking in **Hashtable**.

If u can see diagram the Hashtable applies lock on entire Hashtable but in ConcurrentHashMap it applies lock on segment or u can say this is a bucket of a map. So, in each bucket it will apply the lock not on entire object.

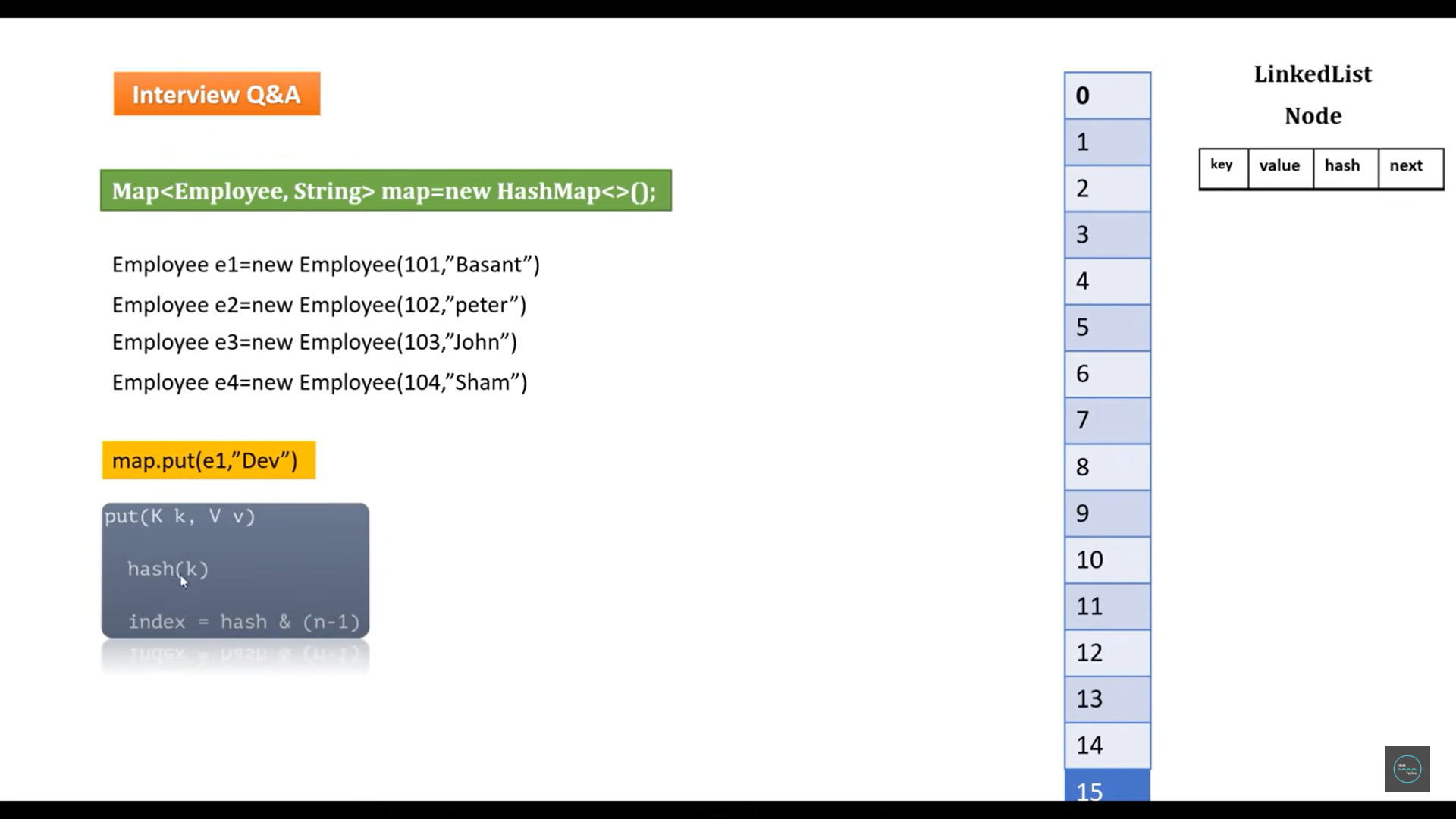
1. **We can Synchronize a HashMap using Collections then why can’t we use that instead using ConcurrentHashMap?**

If we used **Collections.synchronizedMap(map)** it will act as a synchronized Hashtable only where again locking mechanism is different.

1. **How HashMap Internally works?**

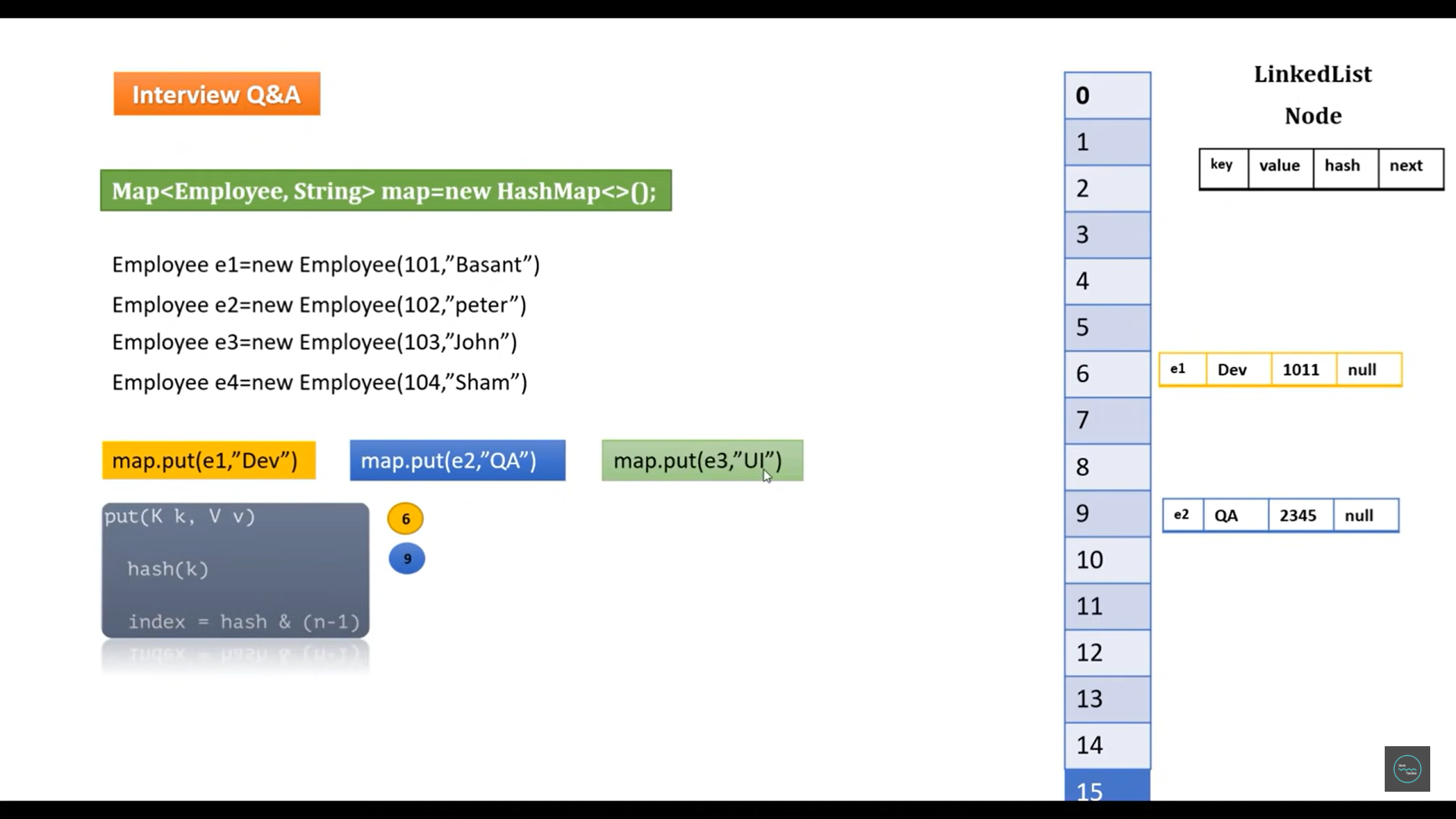
Let’s say I created one map object. When we create a HashMap object internally it creates a bucket structure like this since initial capacity of map is 16 initially it creates 16 buckets. Index begin from 0.each bucket internally uses a linked list. So, in these 16 buckets every bucked is considered as a linked list. This **LinkedList** contains one Node. If u observed the internal structure of Node, it contains **key**, **value**, **hash** and **next**. Which means each bucket can be considered as a linked list. Each linked list or each bucket can have n number of nodes.

Let’s create 4 Employee objects to store into a map.



Now when I use **map.put(e1, “Dev”)** . now how HashMap will identified where I need to keep this entry between 0 to 15. So, what HashMap internally will do when we call put method. Inside put method there is a method called **hash**(**key**) it will take argument as your key, so my key is nothing Employee 1 object so it will pass to the hash function it will evaluate some hash value and then based on the modular operator it will identify the index.

Let’s assume this first entry evaluate index 6 now simply this entry or this node will simply go and store in bucket 6. My bucket 6 is nothing a linked list and it will store one node. Now I am trying to add 2nd object again it will come to the put method and again it will evaluate the hash and then again it evaluates the index. It evaluates the index as 9. then it will simply go and store in my bucket 9. So, we can see again it store key, value, hash and next reference null bcz no next node present in that current bucket.

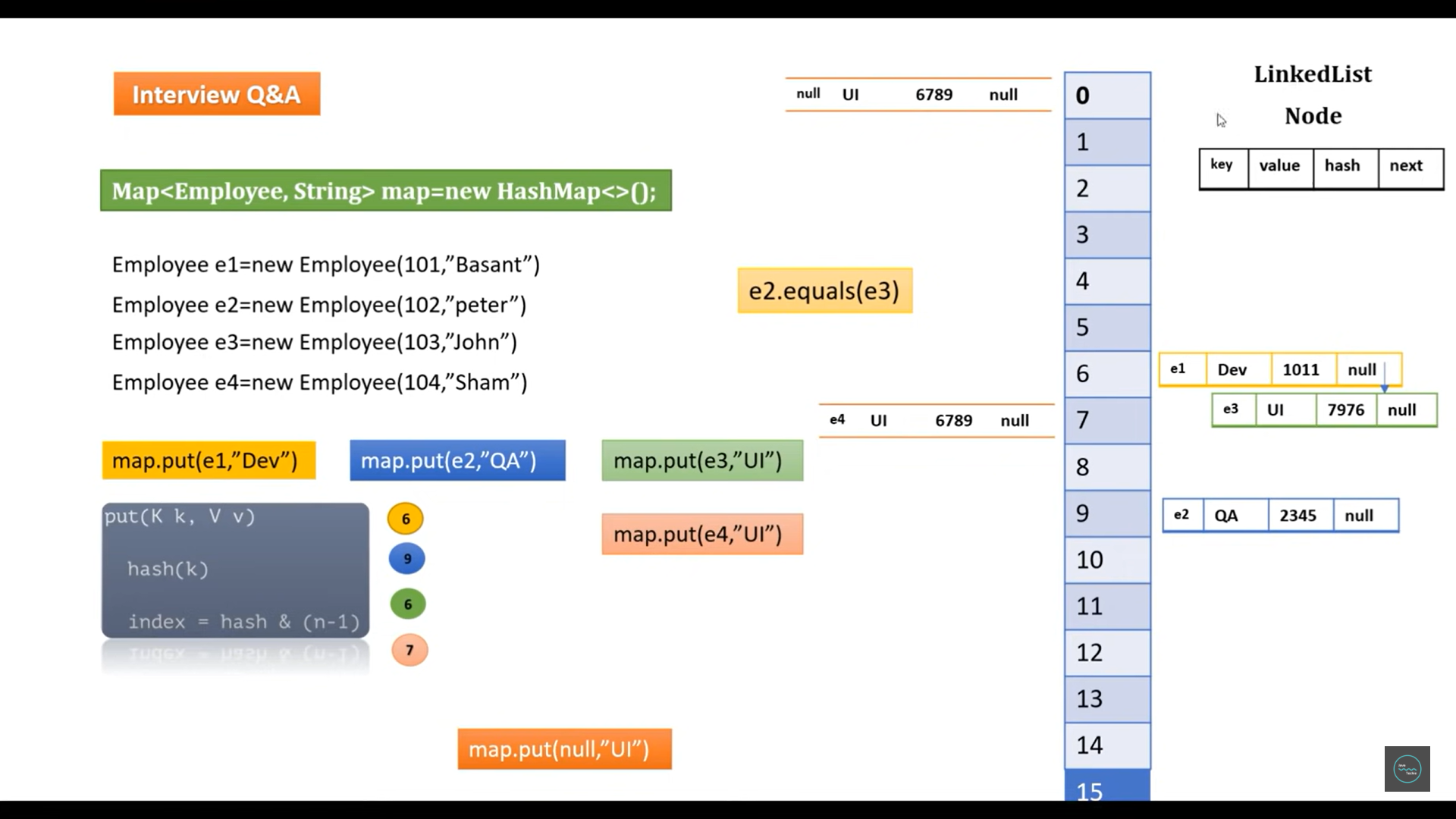


Now let me add 3rd object e3 with some different string. Again, it come to the put method and it evaluate the hash then it find the index coincidently let’s assume it find the same index which is 6. Now if we will go and check there is already one element. **Now if in a same bucket if you find multiple nodes then that concept is called Hashing Collison**.

So, in that Hashing Collison directly my map will not add this entry to the bucket number which is evaluated by this index which is 6. So, what it will internally do since both having the same hash value, so it places on the same bucket immediately map go and check the equals methods **e1.equals(e3)** to check content wise both are different or same. If it finds different, then immediately it will store that entry to the same bucket which is nothing but 6 and now the first entry node next value will not be null it will be the reference of second next node and the second node next would be the null.

If in a same bucket there is a multiple node, then this condition is called Hashing Collison. So, to avoid the hashing Collison map internally use **==** operator to check the reference if both are same reference or different reference. If it found same reference it will just replace if it found different reference then immediately it will go and check the content so, the content between the e1 and e3. e1 equal to e3 if it is true then again it will replace if it found different then it will just add as a next node right to the first node.

Now we have one more element which is **e4** let’s assume it found the index 7. Then that node directly goes to the 7 bucket.



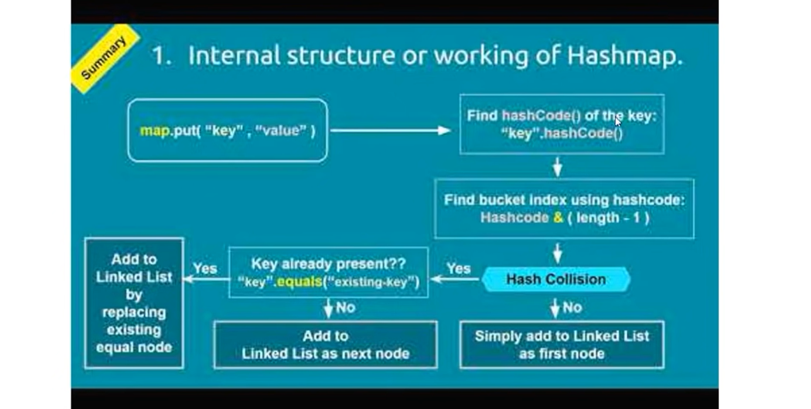
Now interviewer immediately ask you let’s say I pass the key as a **null**. In map I pass key as a null and a value as a null then where it will place. Bcz we can’t evaluate the hash and index based on null. If the key is null then the entry will be added into the 0 bucket.

This is the Internal Implementation of HashMap.

1. **What is the enhancement done in HashMap java 8 ?**

Initially in this HashMap each Bucket is used as a linked list but in certain threshold it will converted to the Balance Tree mechanism. That will not be linked list. Let’s say there is a hashing Collison, in the same bucket they found 5 nodes. Then immediately that linked list will convert to the balance tree and then it will maintain the balanced between the nodes. Now sure much on Balanced tree internal architecture just go and found on google.

Initially it will use LinkedList only but after some threshold it will converted to the balanced tree.

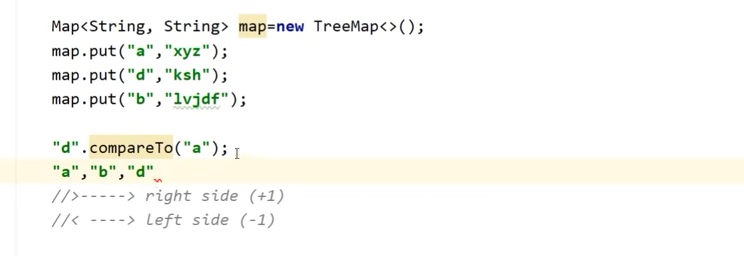


1. **If key is null in HashMap then where that entry will store in map ?**

If the key is null, then the entry will be added into the 0 bucket.

1. **Map Enhancement in Java 8 ?**
2. **How TreeMap internally works?**

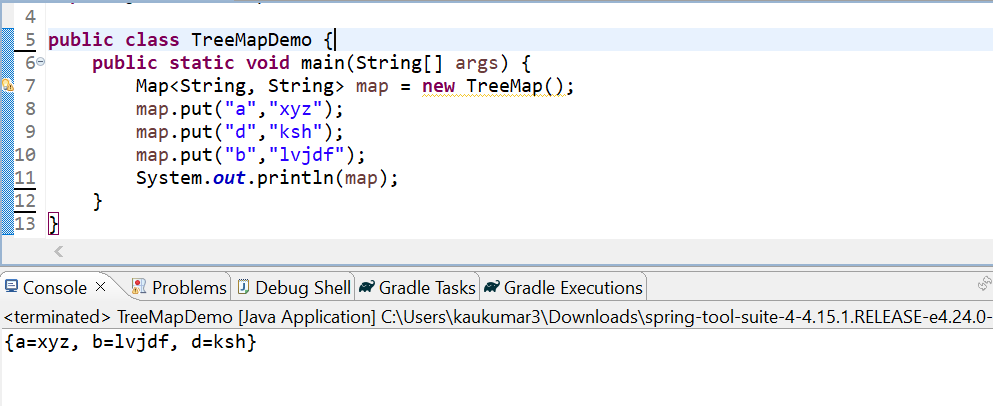
if you use TreeMap any object will add as a key will preserved the default sorting order then how it is doing the default sorting order. Let’s see internal implementation.



Now what happened already one entry exist key as **a** and value as a **xyz** now whenever you will try to add the second entry or second object as a key what tree map internally do it will simply check **“d”.compareTo(“a”).** d is nothing just a second key compared to the first key which is a. now this **compareTo** method will always return integer. now **“d”.compareTo(“a”)** if it is greater than then it will just as to the right side and it will return **+positive** one. If it is less than then it will add to the left side. How it will know that is less than or greater than based on the return type either **+ve** or **-ve**.

So, it will just compare each object or each key if compareTo method return +1 then it will simply add in right side if it will return -1 then it will simply add in left side. So, this is how it will work.

Let’s see one example …



So, we can see its sorting based on the key.

Here b is greater than a so it placed at right side and d is greater than b so it placed at right side.