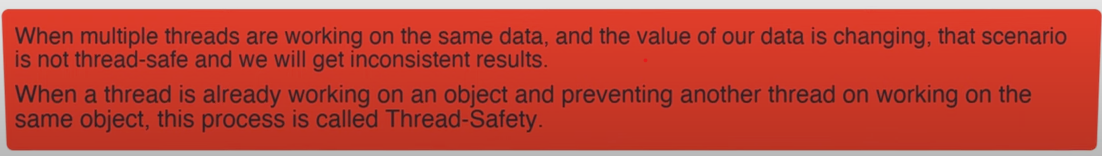
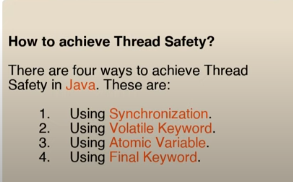


Q)Whats is meant by thread safety?



Q)Major differences bwn collections?





**Q)What is virtual capacity of Array List?**

ArrayList class is a resizable array, present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package.

Difference between an [array](https://www.geeksforgeeks.org/arrays-in-java/) and an [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) in Java, is that the size of an array cannot be modified (i.e. if you want to append/add or remove element(s) to/from an array, you have to create a new array. However, elements can be added/appended or removed from an ArrayList without the need to create a new array.

Whenever an instance of ArrayList in Java is created then by default the capacity of Arraylist is 10. This is virtual capacity.

Note Actual size or Physical capacity will be 0 only.

If we want to increase the virtual capacity. Pass as argumrnt.

ArrayList<Object> ar=new ArrayList<Object>(20);//now VC will be 20 not 10.

Q)**Generics in ArrayList**

* List list = new ArrayList();
* list. add(10);
* list. add("10");
* With Generics, it is required to specify the type of object we need to store.
* List<Integer> list = new ArrayList<Integer>();
* list.add(10);
* list.add("10");// compile-time error.

**Q)Ways to iterate over Array List?**

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

//for Loop

**for**(**int** i=0;i<stuList.size();i++)

System.***out***.println(stuList.get(i));

System.***out***.println("-----------------");

//for each Loop

**for**(String s:stuList)

System.***out***.println(s);

System.***out***.println("-----------------");

//JDK 8 Streams with Lambda

stuList.stream().forEach(ele -> System.***out***.println(ele));

System.***out***.println("-----------------");

//Iterator

Iterator<String> itr=stuList.iterator();

**while**(itr.hasNext())

{

System.***out***.println(itr.next());

}

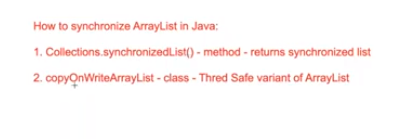
**Q)Given a list , write code to remove the even numbers and display only odd numbers from the list.**

ArrayList<Integer> numbers=**new** ArrayList<Integer>(Arrays.*asList*(1,2,3,4,5,6,7,8,9,10));

numbers.removeIf(num -> num%2==0);

System.***out***.println(numbers);

**Q)How to**[**#Synchronize**](https://www.youtube.com/hashtag/synchronize)**(ThreadSafe) ArrayList in Java | What is CopyOnWriteArrayList class in Java?**

****

//1. Collections.SynchronizedList Method

List<String> nameList=Collections.*synchronizedList*(**new** ArrayList<String>());

nameList.add("java");

nameList.add("python");

nameList.add("ruby");

//for add or remove we dont need explicit synchronization

//to fetch / traverse we need to use explicit synchronization.

**synchronized** (nameList) {

Iterator<String> itr= nameList.iterator();

**while**(itr.hasNext())

System.***out***.println(itr.next());

}

//2. copyOnWriteArrayList- its a class

CopyOnWriteArrayList<String> empList= **new** CopyOnWriteArrayList<String>();

empList.add("arya");

empList.add("hareesh");

empList.add("anitha");

//we dont need any explicit synchronization for any operation. add/remove/traverse

Iterator<String> itr2= empList.iterator();

**while**(itr2.hasNext())

System.***out***.println(itr2.next());

}

# Q)How to Remove Duplicate Elements from an ArrayList using LinkedHashSet & JDK8 Streams?

ArrayList<Integer> numbers=**new** ArrayList<Integer>(Arrays.*asList*(1,1,2,2,33,4,44,4,4,4,55,55,6,6,6,7,7,8,8,9,9,10,10));

//1.Linked Hash Set

LinkedHashSet<Integer> linkedHashSet=**new** LinkedHashSet<Integer>(numbers);//passing list

ArrayList<Integer> withoutDuplicates=**new** ArrayList<Integer>(linkedHashSet);

System.***out***.println(withoutDuplicates);

ArrayList<Integer> markList=**new** ArrayList<Integer>(Arrays.*asList*(1,1,2,2,33,4,44,4,4,4,55,55,6,6,6,7,7,8,8,9,9,10,10));

//2 . JDK-8 Stream

List<Integer> unique=markList.stream().distinct().collect(Collectors.*toList*());

System.***out***.println(unique);

# Q) How to Compare Two ArrayLists in Java?

//1. Sort and then equals

ArrayList<String> l1= **new** ArrayList<String>(Arrays.*asList*("A","B","C","D","F"));

ArrayList<String> l2= **new** ArrayList<String>(Arrays.*asList*("A","B","C","D","E"));

Collections.*sort*(l1);

Collections.*sort*(l2);

**if**(l1.equals(l2))

System.***out***.println("equal");

**else**

System.***out***.println("not equal");

//2.compare the list , findout the additional element

ArrayList<String> l3= **new** ArrayList<String>(Arrays.*asList*("A","B","C","D","F"));

ArrayList<String> l4= **new** ArrayList<String>(Arrays.*asList*("A","B","C","D","E"));

l3.removeAll(l4);

System.***out***.println(l3);

//3.Findout the missing elements in l3

l4.removeAll(l3);

System.***out***.println(l4);//will print which is missing in l3

//4 Find common elements in Array List

ArrayList<String> lang1=**new** ArrayList<String>(Arrays.*asList*("python","java","ruby","C#","js","angular"));

ArrayList<String> lang2=**new** ArrayList<String>(Arrays.*asList*("python","java","ruby","PHP","scala",".net"));

lang1.retainAll(lang2);//returns which is common in both

System.***out***.println(lang1);

# Q) Hash Map

HashMap<K,V>

Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique.

 It is easy to perform operations using the key index like updation, deletion, etc.

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.
* The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

//no order - no indexing

//stores values - Key--Value <K,V>

//Key cannot be duplicate.. if duplicate given, it will override the old value with the new value

//can store n number of null values but only one null key.if 2 null key given, latest value associated will take

//Hash Map is not thread safe -Unsynchronized

//for traversing use iterator, but not directly on the HashMap, use on either Key or value

HashMap<String, String> capitalMap=**new** HashMap<String, String>();

capitalMap.put("India", "New Delhi");

capitalMap.put("USA", "WashingTon DC");

capitalMap.put("UK", "London");

capitalMap.put("UK", "London1");

capitalMap.put(**null**, "Berlin");

capitalMap.put(**null**, "los angales");

capitalMap.put("Russia", **null**);

capitalMap.put("Autria", **null**);

capitalMap.remove("Autria");

System.***out***.println(capitalMap.get("India"));

System.***out***.println(capitalMap.get("UK"));

System.***out***.println(capitalMap.get(**null**));

System.***out***.println(capitalMap.get("Russia"));

//Iterator: over the Keys :by using keySet()

Iterator<String> itr=capitalMap.keySet().iterator();

**while**(itr.hasNext())

{

String key=itr.next();

String value=capitalMap.get(key);

System.***out***.println("Key :"+ key + " value :"+ value );

}

System.***out***.println("----------------------------------");

//Iterator: over the set :by using entrySet()

Iterator<Entry<String,String>> itr1=capitalMap.entrySet().iterator();

**while**(itr1.hasNext())

{

Entry<String, String> entry=itr1.next();

System.***out***.println("Key :"+ entry.getKey() + " value :"+ entry.getValue());

}

System.***out***.println("----------------------------------");

//Iterate hashmap using java8 foreach and lambda:

capitalMap.forEach((k,v) -> System.***out***.println("Key :"+ k + " value :"+ v ));

# Q)How to compare two [#HashMaps](https://www.youtube.com/hashtag/hashmaps) in Java - By Key-Value, By Value and By Keys

HashMap<Integer, String> map1=**new** HashMap<Integer, String>();

map1.put(1, "A");

map1.put(2, "B");

map1.put(3, "C");

HashMap<Integer, String> map2=**new** HashMap<Integer, String>();

map2.put(2, "B");

map2.put(1, "A");

map2.put(3, "C");

HashMap<Integer, String> map3=**new** HashMap<Integer, String>();

map3.put(2, "B");

map3.put(1, "A");

map3.put(3, "C");

map3.put(3, "D");

//1 Compare on the basis of Key value:Use equals method.

//if both key and value are same then true.

System.***out***.println(map1.equals(map2));//true

System.***out***.println(map1.equals(map3 ));//true

//compare hashmap for the same keys: keySet()

System.***out***.println(map1.keySet().equals(map2.keySet()));//true

System.***out***.println(map1.keySet().equals(map3.keySet()));//true

//find out the extra keys

HashMap<Integer, String> map4=**new** HashMap<Integer, String>();

map4.put(2, "B");

map4.put(1, "A");

map4.put(3, "C");

map4.put(4, "D");

//Combine/union the keys from both the maps: using HashSet

//HashSet wont allow duplicate elements..using that property

HashSet<Integer> combineKeys= **new** HashSet<Integer>();

combineKeys.addAll(map4.keySet());

combineKeys.removeAll(map1.keySet());//removes all the keys which are there in map1

//combine keys contains 1,2,3,4 so 1,2,3 which is there in map1 will be removed

System.***out***.println(combineKeys);//only 4 will be present

//4. Compare maps by values

HashMap<Integer, String> map5=**new** HashMap<Integer, String>();

map5.put(1, "A");

map5.put(2, "B");

map5.put(3, "C");

HashMap<Integer, String> map6=**new** HashMap<Integer, String>();

map6.put(4, "A");

map6.put(5, "B");

map6.put(6, "C");

HashMap<Integer, String> map7=**new** HashMap<Integer, String>();

map7.put(1, "A");

map7.put(2, "B");

map7.put(3, "C");

map7.put(4, "C");

//1.Duplicates are not allowed :Using ArrayList

System.***out***.println(**new** ArrayList<>(map5.values()).equals(**new** ArrayList<>(map6.values())));//true

System.***out***.println(**new** ArrayList<>(map5.values()).equals(**new** ArrayList<>(map7.values())));//false

//2.Duplicates are allowed :Using HashSet

System.***out***.println(**new** HashSet<>(map5.values()).equals(**new** HashSet<>(map6.values())));//true

System.***out***.println(**new** HashSet<>(map5.values()).equals(**new** HashSet<>(map7.values())));//true

# Q) How Hash Map works internally?

-> Hashmap works on the basis of Hashing.

->Java uses the hashCode() method which is coming from Object Class

->In each Node out of default 16 nodes in hashmap will maintain a hash,key,value and will maintain a next pointer also.

->Hashcode will be calculated for the key we have written.

->Multiple objects can have the same hashcode and the node will be same for those objects.

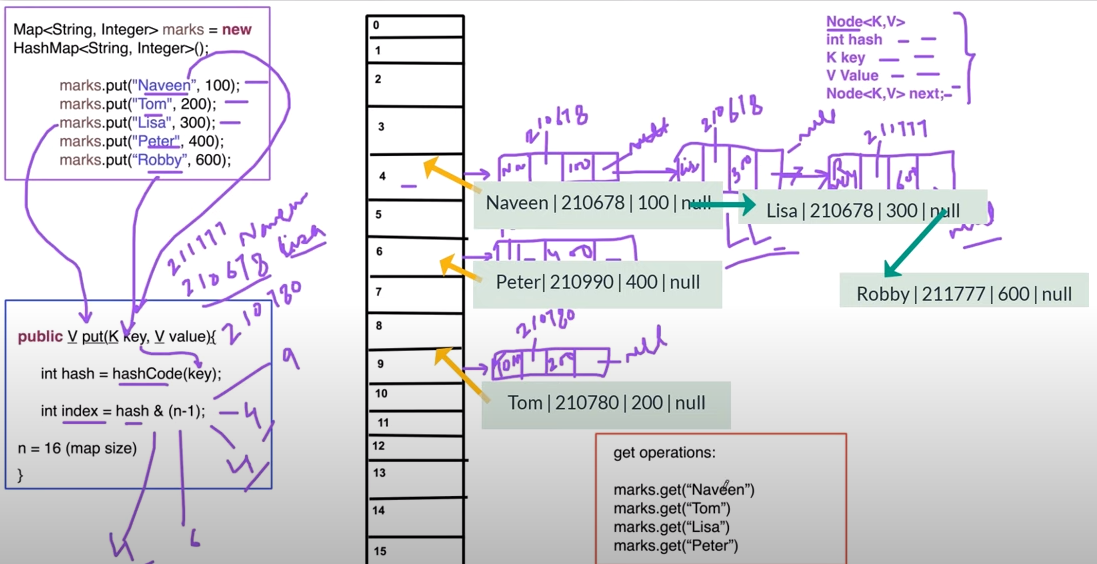
->Even if hashcodes are different ,the nodes can be same for different objects.

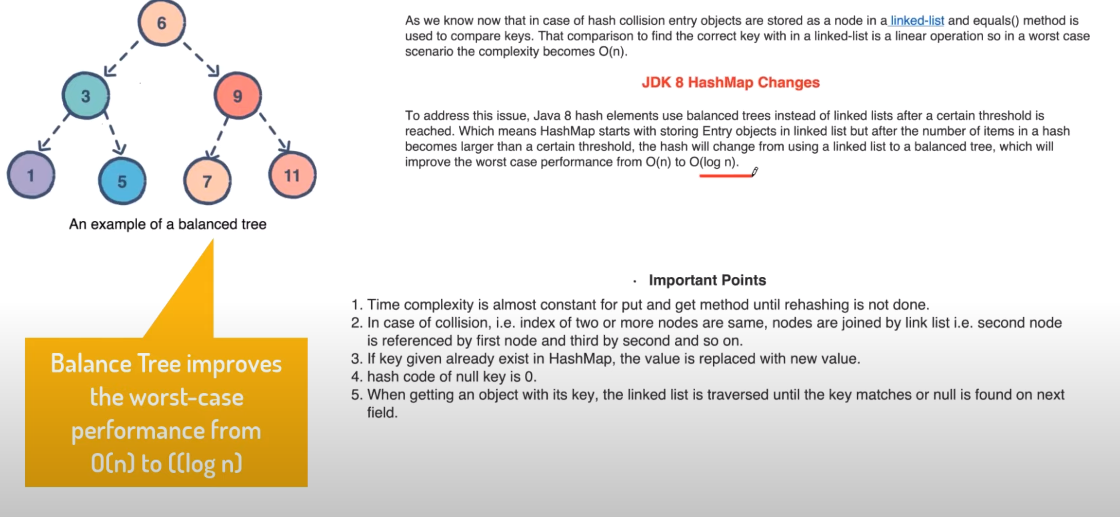
->So internally Hash Map will behave as a linked list.

->When same index is getting for different objects, then that condition is called as collision.

->We can pass one key as null as well.Note that Hashcode for null keys are always 0,so index will be also 0 in that case.(Interview question).







# Q) Different ways of creating HashMap in Java

//1. Using HashMap class

HashMap<String, String> map1=**new** HashMap<String, String>();

Map<String,String> map2=**new** HashMap<>();

//2. Static way :Static HashMap

System.***out***.println(HashMapInitialisation.*marksMap*.get("A"));

//3.Immutable map with only one single entry

Map<String,Integer> map3=Collections.*singletonMap*("test", 100);

System.***out***.println(map3.get("test"));

//map3.put("abc", 200);//UnsupportedOperationException

//4 . JDK 8

//creating one 2D array of Strings using Stream and collecting in the form Map

Map<String, String> map4=Stream.*of*(**new** String [][] {

{"Tom","A Grade"},

{"Hareesh","A+ Grade"}

}).collect(Collectors.*toMap*(data->data[0], data->data[1]));

System.***out***.println(map4.get("Hareesh"));

map4.put("Arya", "A++ grade");

System.***out***.println(map4.get("Arya"));

//using SimpleEntry :mutable map

Map<String,String> map5=Stream.*of*(

**new** AbstractMap.SimpleEntry<>("Naveen","Java"),

**new** AbstractMap.SimpleEntry<>("Hareesh","Python")

).collect(Collectors.*toMap*(Map.Entry::getKey, Map.Entry::getValue));

System.***out***.println(map5.get("Hareesh"));

map5.put("Arya", "Cpp");

System.***out***.println(map5.get("Arya"));

//JDK 1.9

//EmptyMap

Map<String,String> emptyMap=Map.*of*();

//emptyMap.put("Hareesh", "Elavally");

//System.out.println(emptyMap.get("Hareesh"));//UnsupportedOperationException

//singleton map

Map<String,String> singletonMap=Map.*of*("k1","v1");

System.***out***.println(singletonMap.get("k1"));

//singletonMap.put("k2", "v2");//UnsupportedOperationException

//Multi value Map : max of 10 key value pairs can be stored

Map<String,String> multivalueMap=Map.*of*("k1","v1","k2","v2","k3","v3");

System.***out***.println(multivalueMap.get("k3"));

//multivalueMap.put("K4", "v4");//UnsupportedOperationException

//using Map.ofEntries : no limitations on key and value pairs

//immutable

Map<String, Integer> map7=Map.*ofEntries*(

**new** AbstractMap.SimpleEntry<>("A",100),

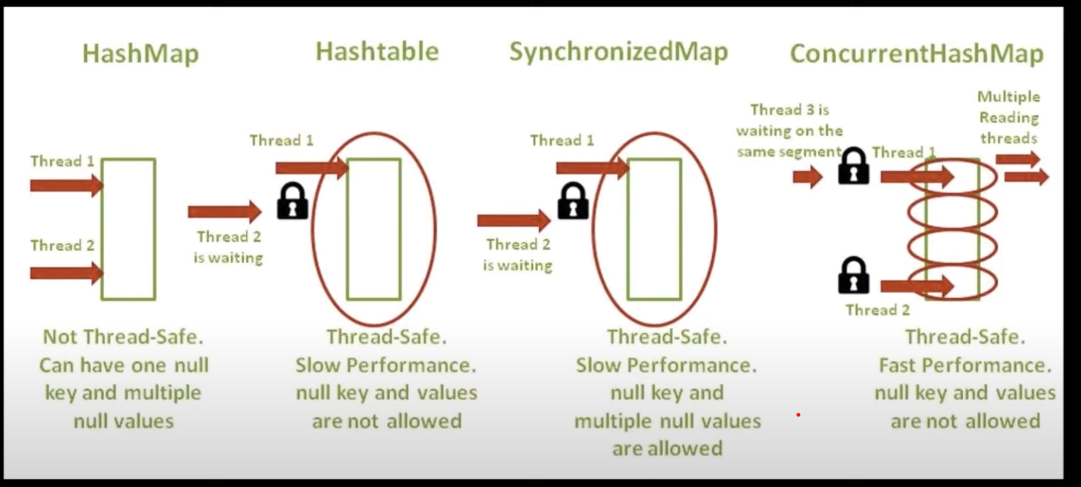
**new** AbstractMap.SimpleEntry<>("B",200)

);

System.***out***.println(map7.get("A"));

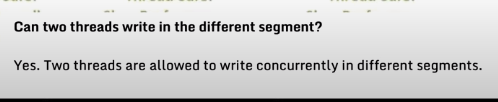
# //map7.put("C", 300);//UnsupportedOperationException

# Q)Difference b/w ConcurrentHashMap & SynchronizedMap in Java || (HashTable vs HashMap)

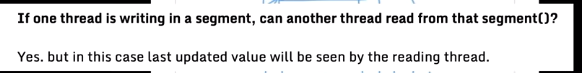


* Segment level lock is applied for concurrent hashmap not on object level.
* The object will be divided into 16 segments.









//synchronizedMap method in collection class

Map<String,String> map1=**new** HashMap<String, String>();

map1.put("1", "Hareesh");

map1.put("2", "Arya");

map1.put("3", "Mahendran");

//create Synchronized Map

Map<String,String> syncMap=Collections.*synchronizedMap*(map1);

System.***out***.println(syncMap);

//ConcurrentHashMap:It doesnt throw any ConcurrentModificationException

ConcurrentHashMap<String, String> concMap= **new** ConcurrentHashMap<String, String>();

concMap.put("A", "Java");

concMap.put("B", "Python");

concMap.put("C", "Angular");

System.***out***.println(concMap.get("A"));

System.***out***.println(concMap);

# Q) How to convert HashMap to ArrayList in Java?

HashMap<String, Integer> compMap=**new** HashMap<String, Integer>();

compMap.put("Google", 10000);

compMap.put("Walmart", 20000);

compMap.put("Amazon", 30000);

compMap.put("facebook", 40000);

compMap.put("cisco", 70000);

//converting keys into 1 ArrayList and values to another

//convert Hashmap keys into ArrayList

List<String> compNamesList=**new** ArrayList<String>(compMap.keySet());

System.***out***.println(compNamesList);

**for**(String t:compNamesList)

System.***out***.println(t);

System.***out***.println("----------------------------");

//convert Hashmap values into ArrayList

List<Integer> empCount=**new** ArrayList<Integer>(compMap.values());

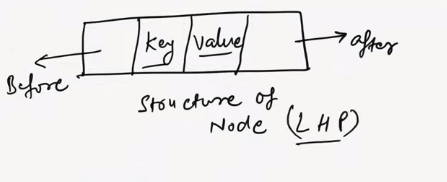
System.***out***.println(empCount);

**for**(Integer i:empCount)

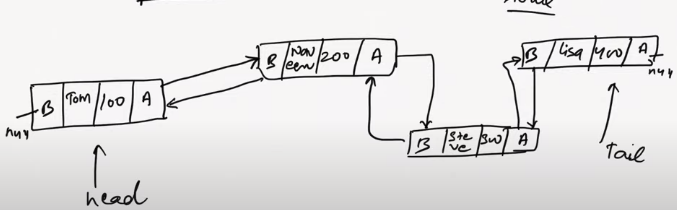
System.***out***.println(i);

# Q)linkedhashmap vs hashmap in Java || How LinkedHashMap works internally?

* LinkedHashMap extends HashMap class
* LinkedHashMap maintains insertion order
* It stores the value in a node and the node is defined like a doubly linked list,
* It is not synchronized.
* One null key and multiple null values are allowed.

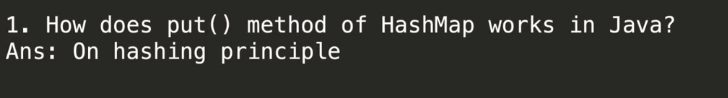


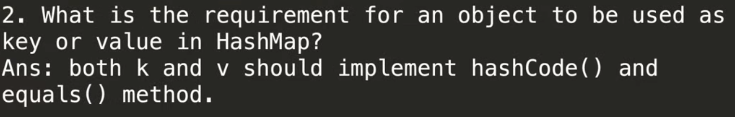
* It will be divided into 16 segments by default.
* The concept of hashing is same as hash map.
* Every segment will behave like a node.



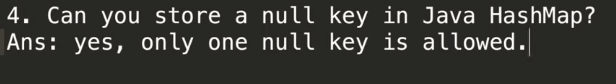
* In the insertion order the elements will be linked to each other.The newly added element will have the previous element address and the previous element will have the current element address..like that.

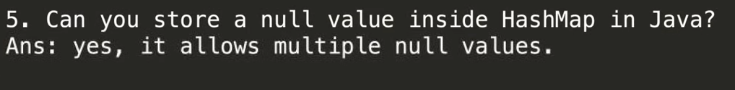
# Q)HashMap Interview Questions





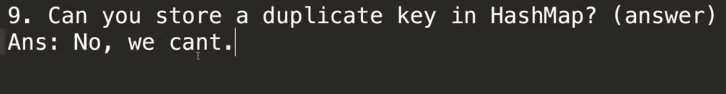






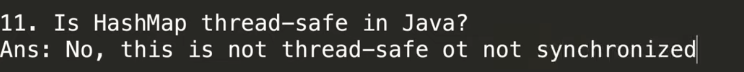
7) Which data structure hashmap represents?

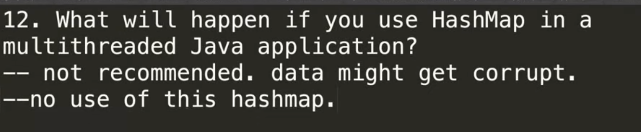
Linked List and Binary Tree.



10) Can we store duplicate value in Hashmap?

No We cant. If we are doing like that, we will be getting latest value.





Multiple threads will try to modify the data at the same time.so if another thread is accessing the data it may or may not get the latest value.

