Collection Framework

Collections

- Introduction
- Limitation of Array
- Advantages of Collections
- Collection vs Collections
- Collection Interfaces
- Set
- List
- Map
- Iterators
- List Iterators
- Wrapper Classes
- Boxing, Autoboxing, Unboxing
- Syncronization

Concurrent Collections

- ConcurrentHashMap
- CopyOnWriteArrayList
- CopyOnWriteArraySet

Limitations of Array	Advantages of collections
Fixed size	Growable in nature
Stores only homogeneous data	Stores any type of data
Performance is better	Performance is low
No any Underlying data structure	Underlying Data structure available
Use Java.lang pakage	Use java.util.* package
No any in-build method to read, remove, add etc.	In-build method supports like add, remove, get etc.
No any iterator avaiable	Iterator and ListIterator available

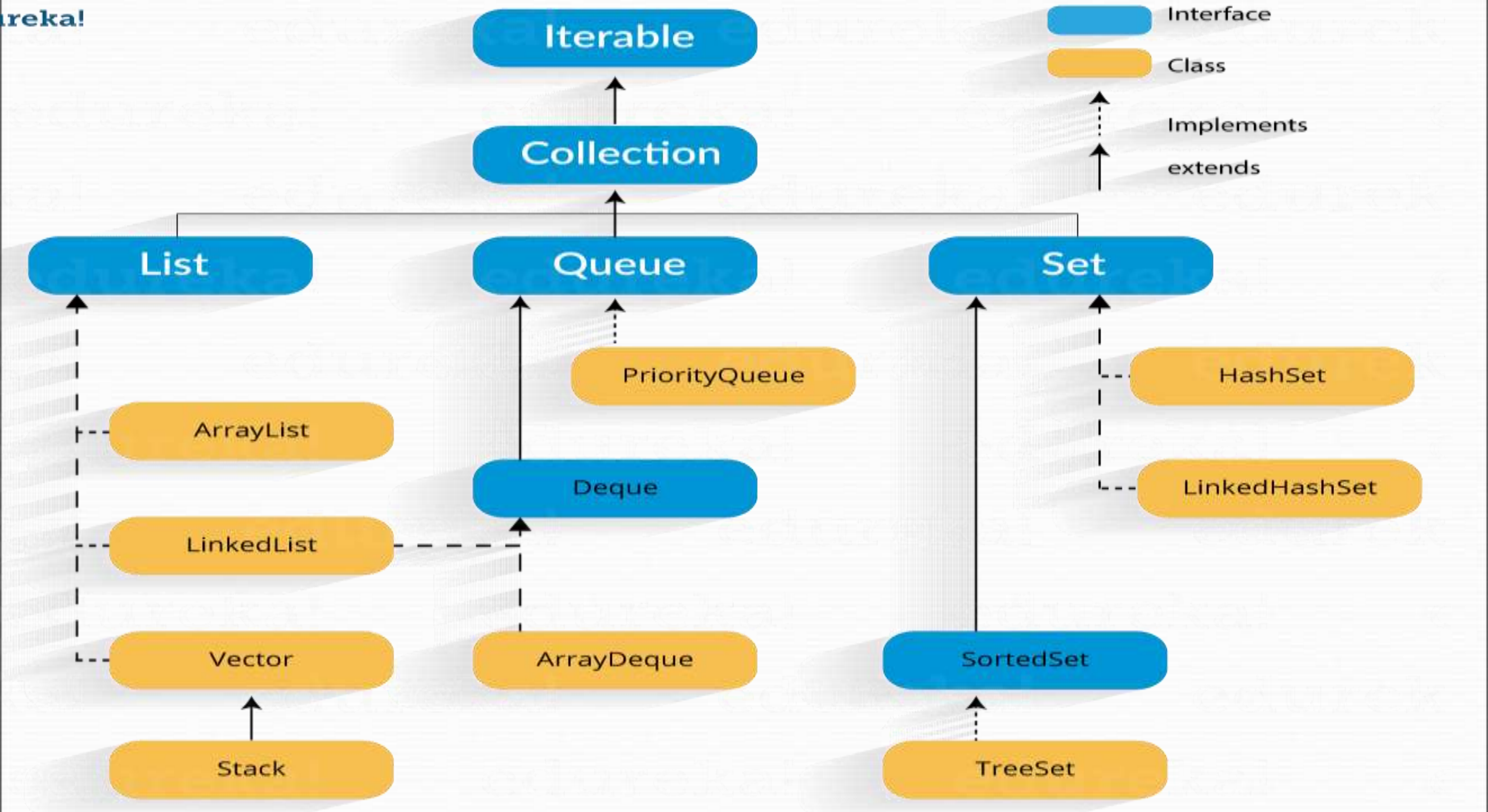
- Collection

  - A group of individual objects as a single entity.
  - Example – Set , List

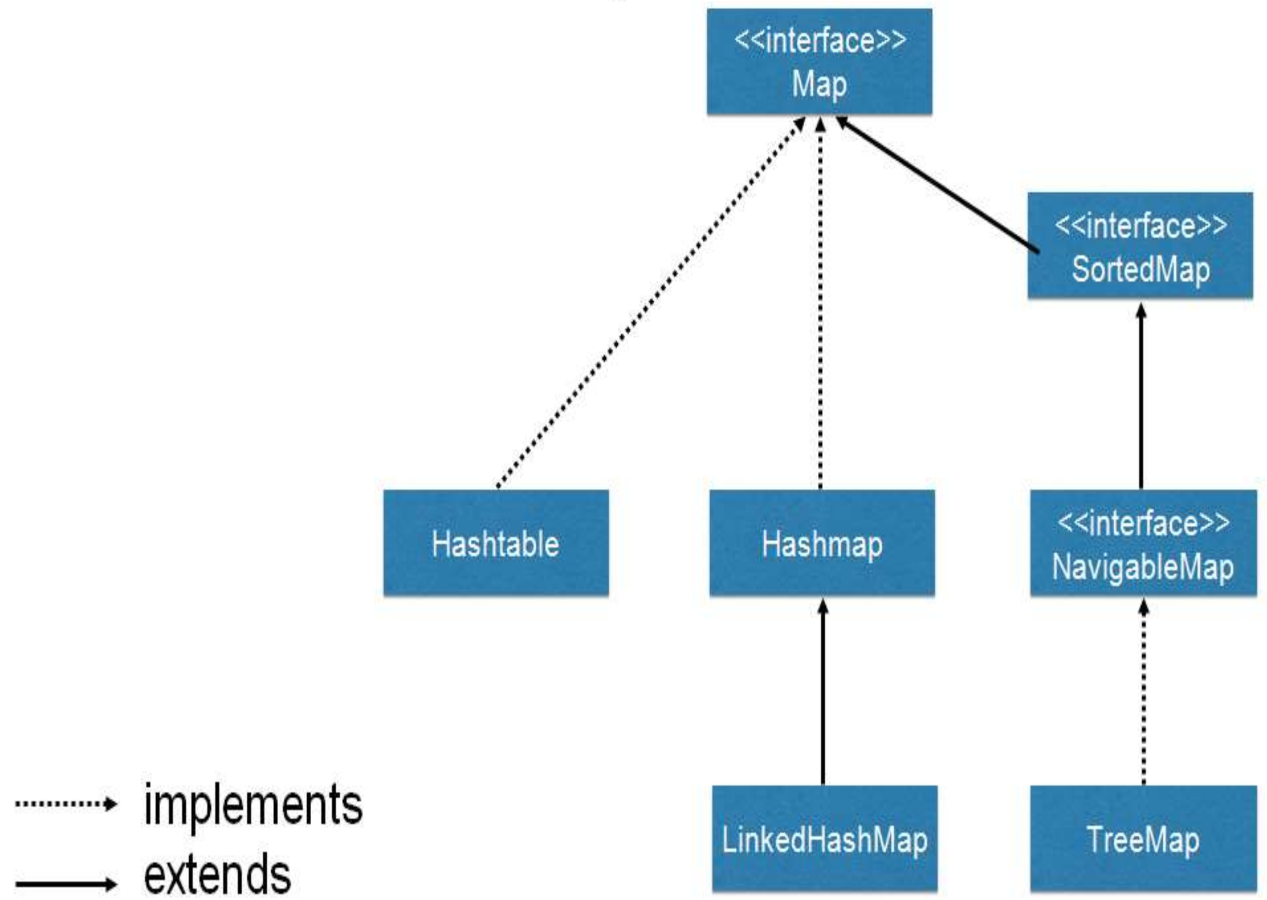
- Collections

  - Collections is an utility class present in java.util package, to define several utility methods for collections.
  - Example – Collections.sort();

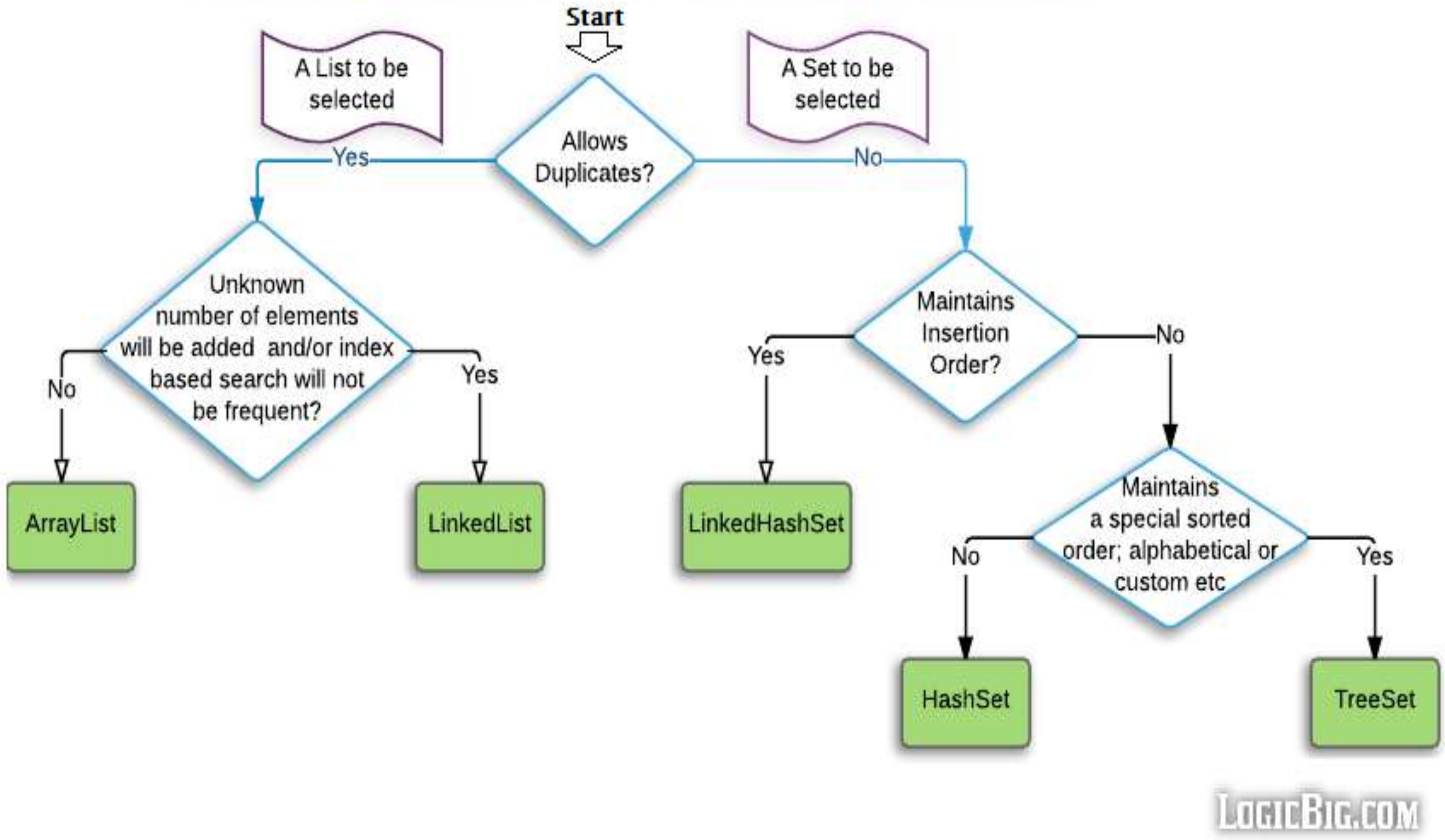
- Collection Hierarchy



Map Interface



Choosing A Collection Implementation


Start
Allows Duplicates?
A List to be selected
Yes
A Set to be selected
No
Unknown number of elements will be added and/or index based search will not be frequent?
No
Yes
ArrayList
LinkedList
Maintains Insertion Order?
Yes
No
LinkedHashSet
Maintains a special sorted order; alphabetical or custom etc
No
Yes
HashSet
TreeSet
LogicBig.com

Set Interface

- Duplicate objects/elements are not allowed.
- Insertion order is not preserved
- In build methods – add, remove
- Classes of set interface
  - HashSet – Insertion order is not preserved (DS –Hashtable)
  - LinkedHashSet – Insertion order is preserved (DS- HashTable + linkedList)
  - SortedSet – Default natural sorting order
  - TreeSet – Sorting objects (DS – Balanced Tree)

HashSet

- Data Structure – HashTable
- Insertion order not preserved
- Duplicate elements are not allowed
- Syntax :
  - Set hs =new HashSet();
  - HashSet hs= new HashSet();
- If you want to store only specific data types i.e. type safe

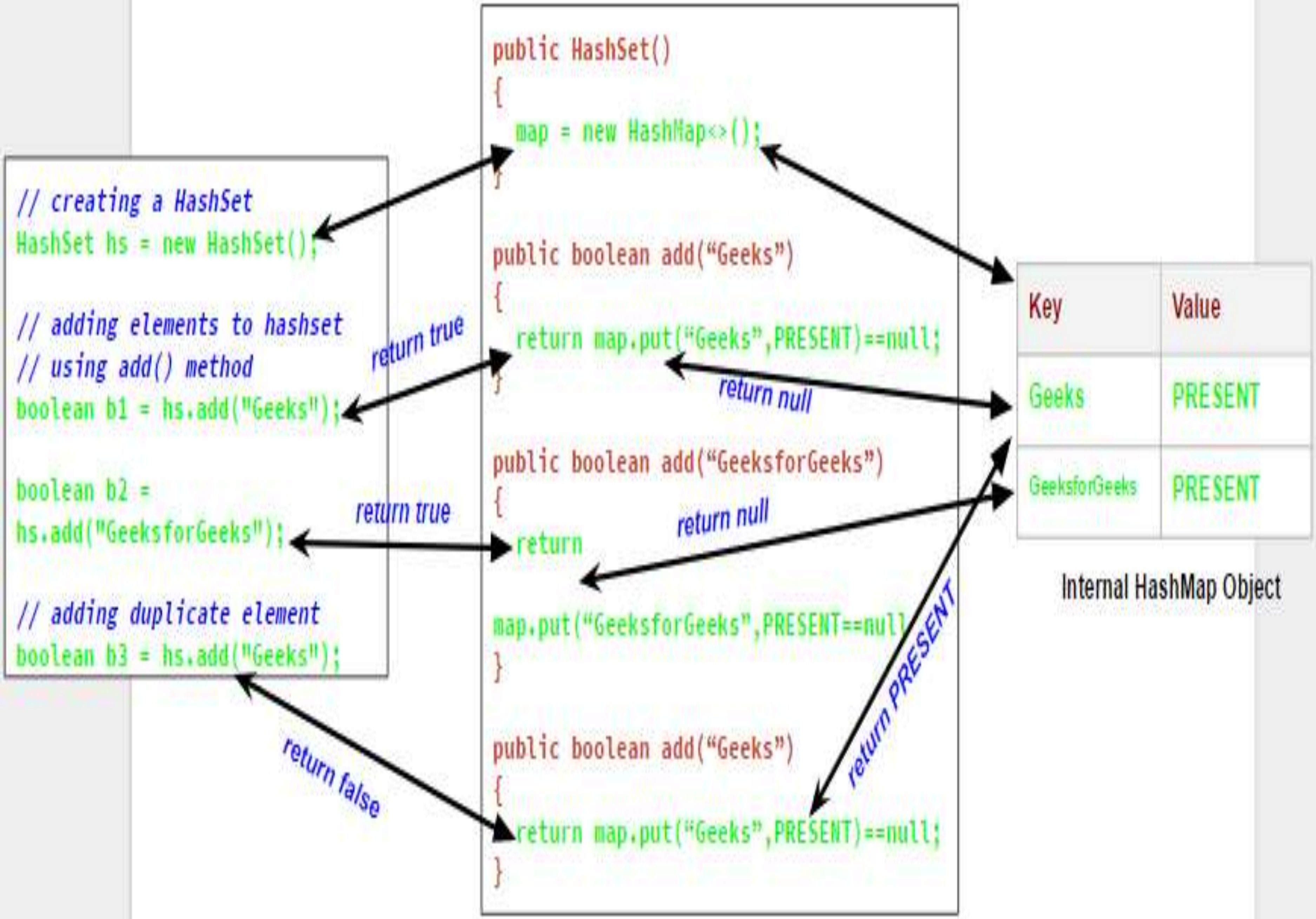
  Set<String> hs =new HashSet();
- Methods:
  - add(object) , remove(object) , size(), contains(object)
- HashSet implements **Serializable** and **Cloneable** interfaces
- HashSet is not synchronized.
- Default initial capacity is 16 and the default load factor is 0.75
- Iterator you can use for Iterating or enhanced for loop.

Internal working of a HashSet:

- All the classes of Set interface internally backed up by Map. HashSet uses HashMap for storing its object internally. Value in HashMap we need a key-value pair, but in HashSet, we are passing only one value.

Storage in HashMap:

- Actually the value we insert in HashSet acts as a key to the map Object and for its value, java uses a constant variable. So in key-value pair, all the values will be the same.



```
// creating a HashSet
HashSet hs = new HashSet();

// adding elements to hashset
// using add() method
boolean b1 = hs.add("Geeks");

boolean b2 =
hs.add("GeeksforGeeks");

// adding duplicate element
boolean b3 = hs.add("Geeks");
```

```
public HashSet()
{
  map = new HashMap<>();
}

public boolean add("Geeks")
{
  return map.put("Geeks",PRESENT)==null;
}

public boolean add("GeeksforGeeks")
{
 return

map.put("GeeksforGeeks",PRESENT==null
}

public boolean add("Geeks")
{
 return map.put("Geeks",PRESENT)==null;
}
```

Key	Value
Geeks	PRESENT
GeeksforGeeks	PRESENT

Internal HashMap Object

- Initial capacity :
  - The initial capacity means the number of buckets when hashtable (HashSet internally uses hashtable data structure) is created.
- Load factor :
  - The load factor is a measure of how full the HashSet is allowed to get before its capacity is automatically increased

- **Example:** If internal capacity is 16 and the load factor is 0.75 then the number of buckets will automatically get increased when the table has 12 elements in it.
- Time Complexity of HashSet operations :
  - add, remove and look-up (contains method) operation of HashSet takes **O(1)** time

LinkedHashSet

- Data Structure – HashTable
- Insertion order is preserved
- Duplicate elements are not allowed
- Syntax :
  - Set hs =new LinkedHashSet();
  - HashSet hs= new LinkedHashSet();
- If you want to store only specific data types i.e. type safe

  Set<String> hs =new LinkedHashSet();
- Methods:
  - add(object) , remove(object) , size(), contains(object)
- LinkedHashSet implements **Serializable** and **Cloneable** interfaces
- LikedHashSet is not synchronized.
- Default initial capacity is 16 and the default load factor is 0.75
- Iterator you can use for Iterating or enhanced for loop.

TreeSet

- Data Structure – Balanced Tree
- Duplicate elements are allowed
- Default sorting order is ascending. (Based on dictionary sort)
- Syntax :
  - Set hs =new TreeSet();
- Methods:
  - add(object) , remove(object) , size(), contains(object)

List Interface

- Duplicate objects/elements are allowed.
- Insertion order is preserved
- In build methods – add, get,remove, size
- Classes of set interface
  - ArrayList– Insertion order is not preserved (DS –Resizable)
  - LinkedList – Insertion order is preserved (DS- LinkedList)
  - Vector – Synchronization default (Legecy classes)

ArrayList

- Data Structure – Resizable/Growable
- Index based list
- Insertion order is preserved by default
- Duplicate elements are allowed
- Syntax :
  - List list =new ArrayList();
  - ArrayList list= new ArrayList();
- If you want to store only specific data types i.e. type safe

  List<String> list =new ArrayList();
- Methods:
  - add(object) , get(index), remove(object) , size(), contains(object)
- ArrayList implements **RandomAccess,Serializable,Cloneable** interfaces
- ArrayList is not synchronized.
- Default initial capacity is 10
- ListIterator - you can use for Iterating or enhanced for loop.

➢Internal working of a ArrayList:

- ➢Internally as ArrayList uses as Object[] Array which is an array objects.
- ➢Grow size by 50%.
- ➢While removing any element then shifting operations happened.

Performance of ArrayList
The time complexity of the common operations in ArrayList java.

➢**add():** For adding a single element O(1) . Adding n element in array list takes O(n).

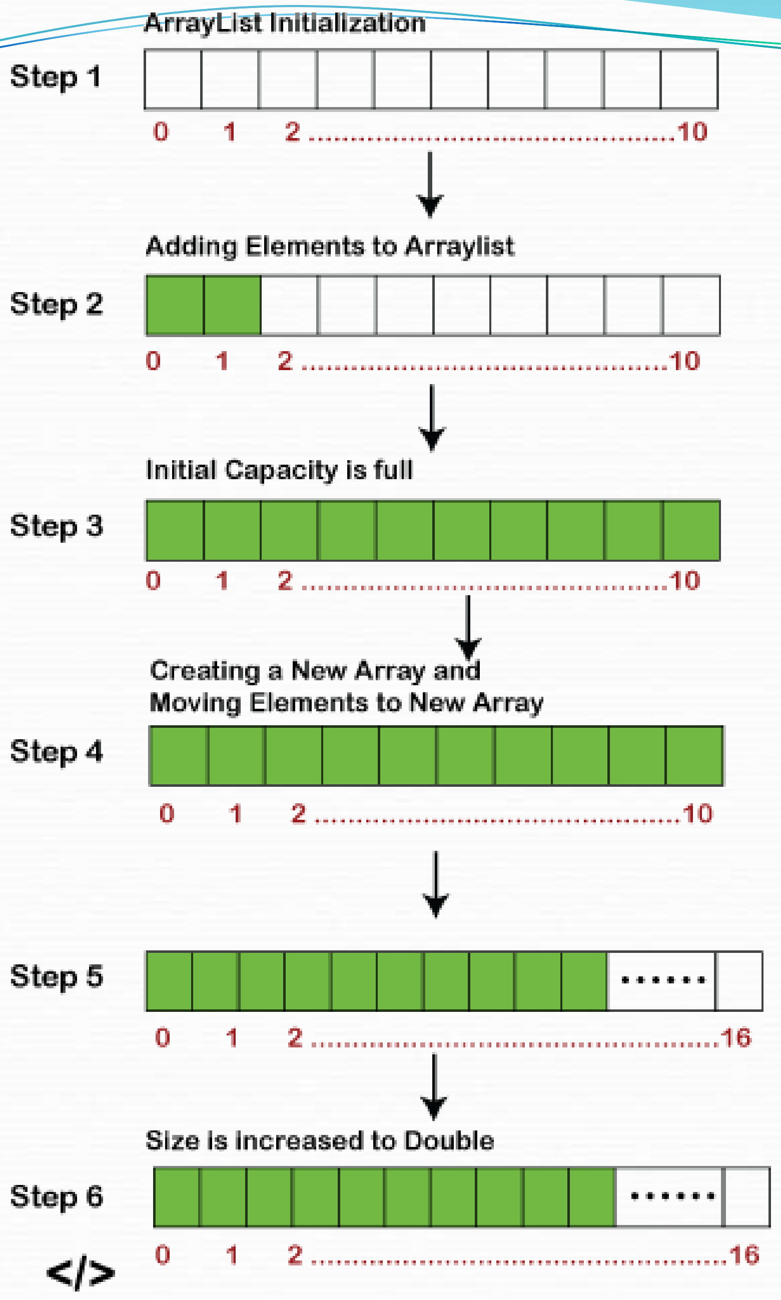
➢**add**(index, element)**:** adding element in particular index in average runs in O(n) time

➢**get():** is always a constant time O(1) operation

➢**remove():** runs in linear O(n) time. We have to iterate the entire array to find the element fit for removal

➢**indexOf():** It runs over the whole array iterate through each and every element worst case will be the size of the array n .so, it requires O(n) time

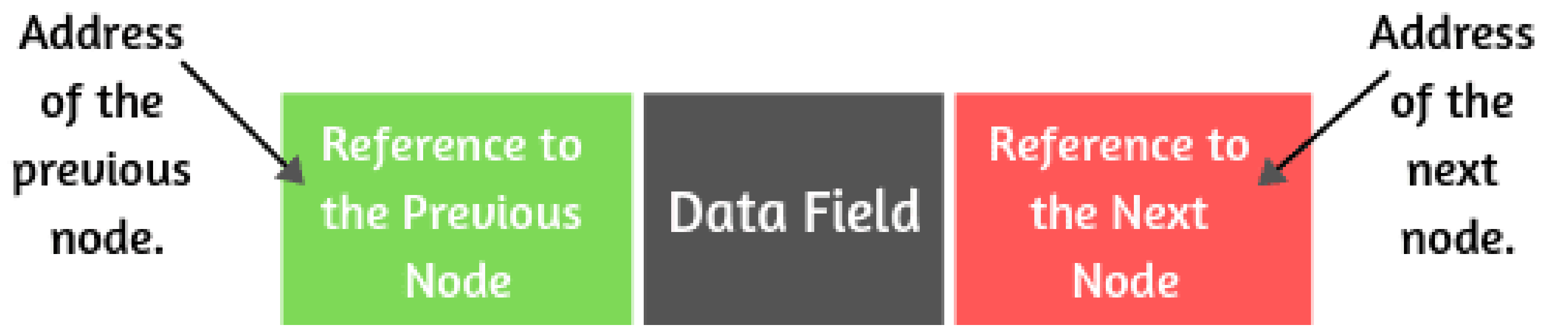
➢**contains():** implementation is based on indexOf(). So it will also run in O(n) time
➢The **size**, **isEmpty**, **set**, **iterator**, and **listIterator** operations run in constant time O(1)


ArrayList Initialization
Step 1
0 1 2 10
Adding Elements to Arraylist
Step 2
0 1 2 10
Initial Capacity is full
Step 3
0 1 2 10
Creating a New Array and
Moving Elements to New Array
Step 4
0 1 2 10
Step 5
0 1 2 16
Size is increased to Double
Step 6
0 1 2 16

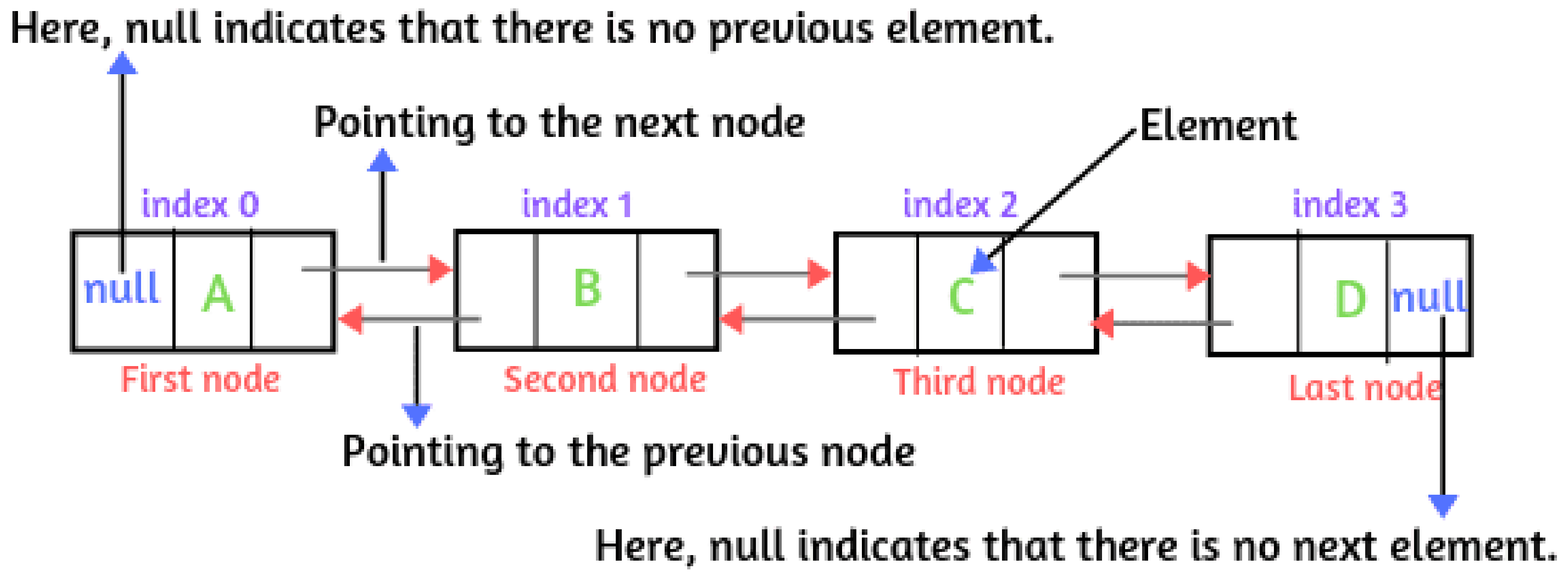
LinkedList:

- Data Structure – Doubly linked list
- Stores address of previous node, next node and elements.
- Insertion order is preserved by default
- Duplicate elements are allowed
- Syntax :
    - List list =new ArrayList();
    - ArrayList list= new ArrayList();
- If you want to store only specific data types i.e. type safe

  List<String> list =new ArrayList();
- Methods:
    - add(object) , get(index), remove(object) , size(), contains(object)
- ArrayList implements **RandomAccess,Serializable,Cloneable** interfaces
- ArrayList is not synchronized.
- Default initial capacity is 10
- ListIterator - you can use for Iterating or enhanced for loop.


Address of the previous node.
Reference to the Previous Node
Data Field
Reference to the Next Node
Address of the next node.

Representation of Java LinkedList Node

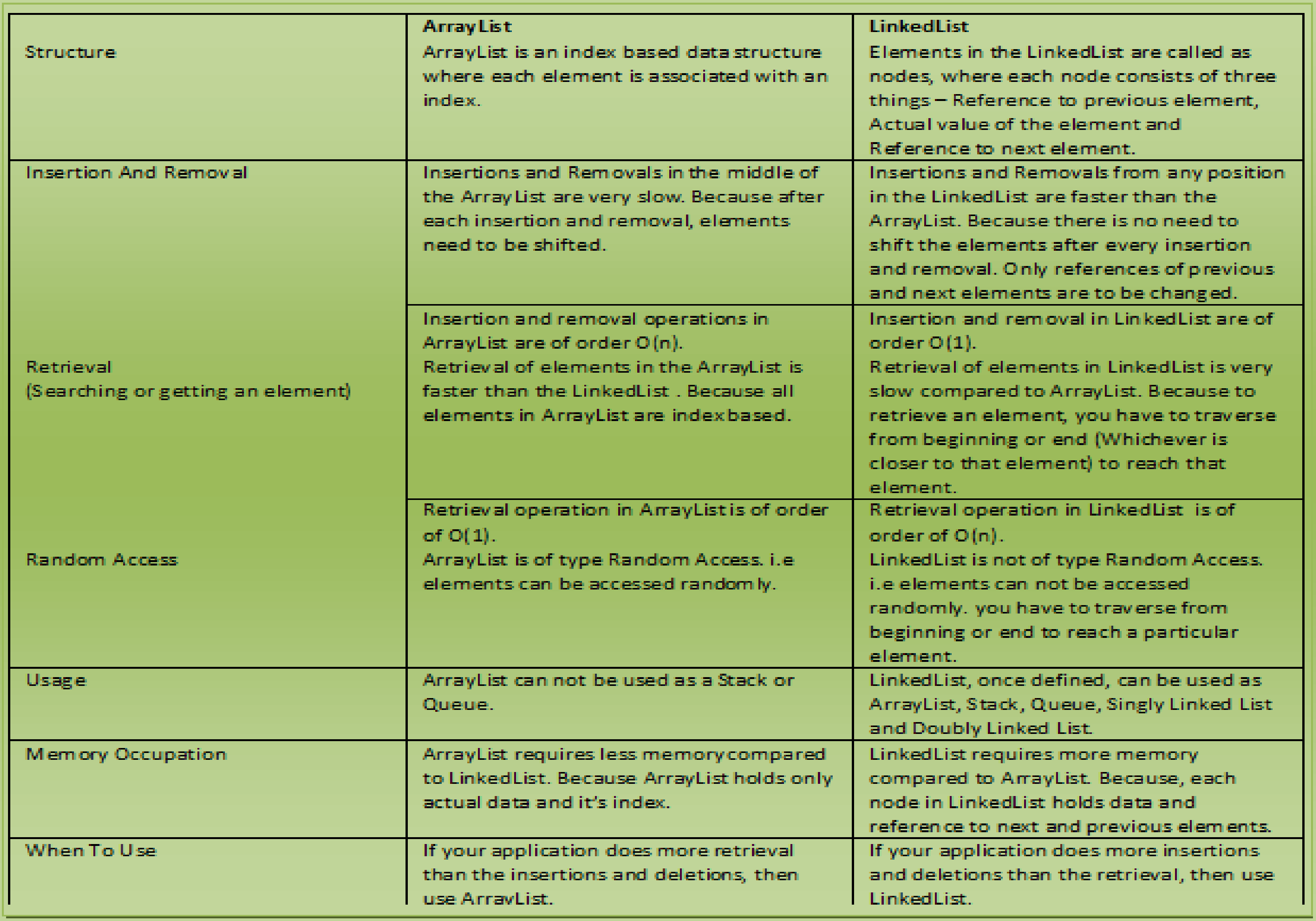

Here, null indicates that there is no previous element.
Pointing to the next node
Element
index 0
index 1
index 2
index 3
null
A
B
C
D
null
First node
Second node
Third node
Last node
Pointing to the previous node
Here, null indicates that there is no next element.

A array representation of linear Doubly LinkedList in Java

Vector:

- Data Structure – Growable/Resizable
- Vector is by default Syncronized
- Duplicate elements are allowed
- Syntax :
  - List list =new ArrayList();
  - ArrayList list= new ArrayList();
- If you want to store only specific data types i.e. type safe

  List<String> list =new ArrayList();
- Methods:
  - add(object) , get(index), remove(object) , size(), contains(object)
- ArrayList implements **RandomAccess,Serializable,Cloneable** interfaces
- ArrayList is not synchronized.
- Default initial capacity is 10
- ListIterator - you can use for Iterating or enhanced for loop.



Structure	**ArrayList** ArrayList is an index based data structure where each element is associated with an index.	**LinkedList** Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element.
Insertion And Removal	Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted.	Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references of previous and next elements are to be changed.
Retrieval (Searching or getting an element)	Insertion and removal operations in ArrayList are of order O(n). Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based.	Insertion and removal in LinkedList are of order O(1). Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element.
Random Access	Retrieval operation in ArrayList is of order of O(1). ArrayList is of type Random Access. i.e elements can be accessed randomly.	Retrieval operation in LinkedList is of order of O(n). LinkedList is not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element.
Usage	ArrayList can not be used as a Stack or Queue.	LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List.
Memory Occupation	ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it's index.	LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and previous elements.
When To Use	If your application does more retrieval than the insertions and deletions, then use ArrayList.	If your application does more insertions and deletions than the retrieval, then use LinkedList.

Map:

- Data Structure – HashTable
- Stores <key, value> pair
- Duplicate key is not allowed
- Duplicate value is allowed
- HashMap may have one null key and multiple null values.
- HashMap is non synchronized.
- HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75

- Syntax :
  - Map<Integer, String> map=new HashMap();
- Classes :
- HashMap
- HashTable

HashMap:

- Data Structure – HashTable
- Store Key , value
- Syntax :
- Map<Integer, String> map=new HashMap();
- Methods:
- put(k,v) , get(k), getKey(), containsKey(k), containsValue()
- EntrySet<key,value>

➢**equals():** It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.

➢**hashCode():** This is the method of the object class. It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map. Hash code of null Key is 0.

➢**Buckets:** Array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.

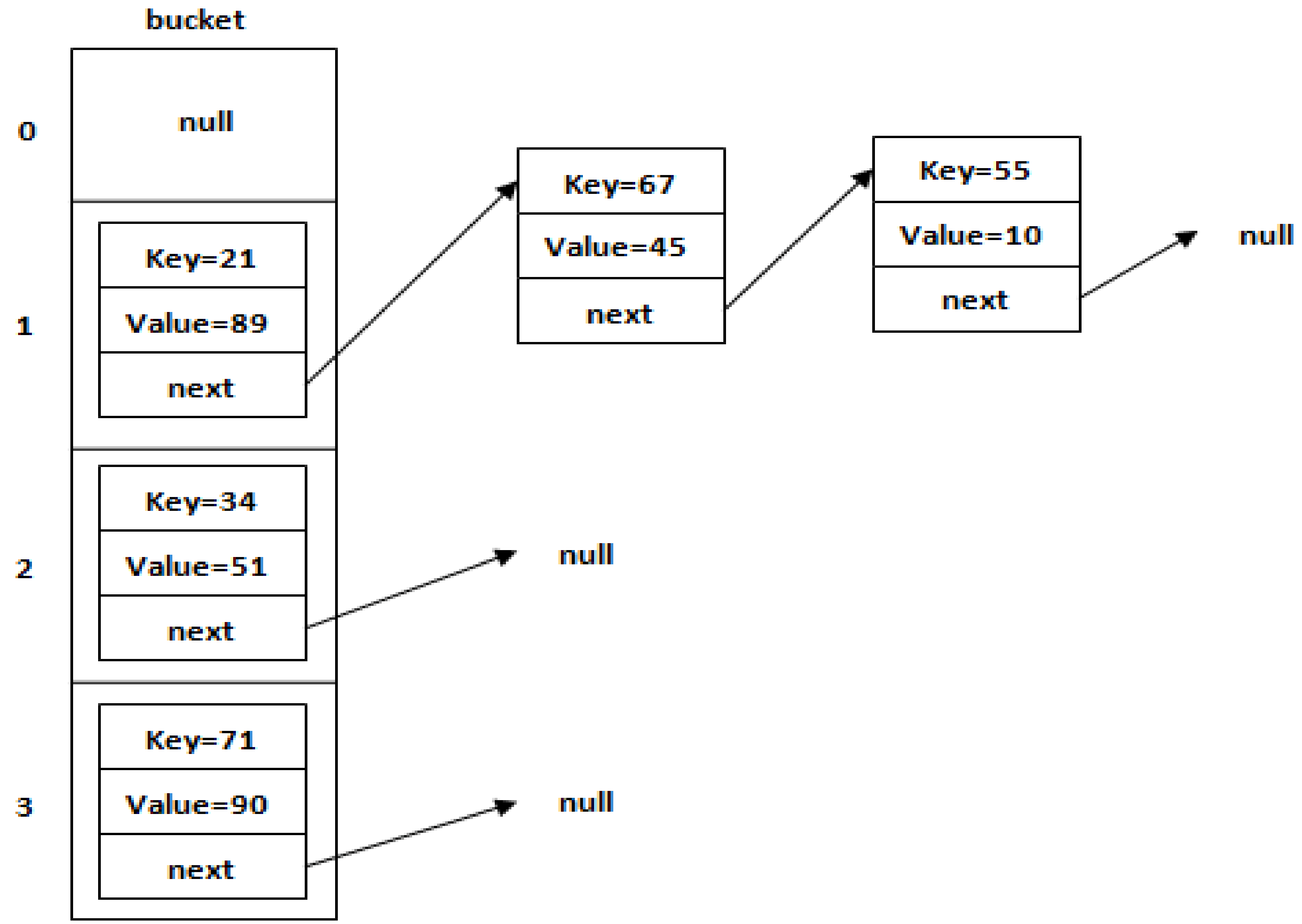
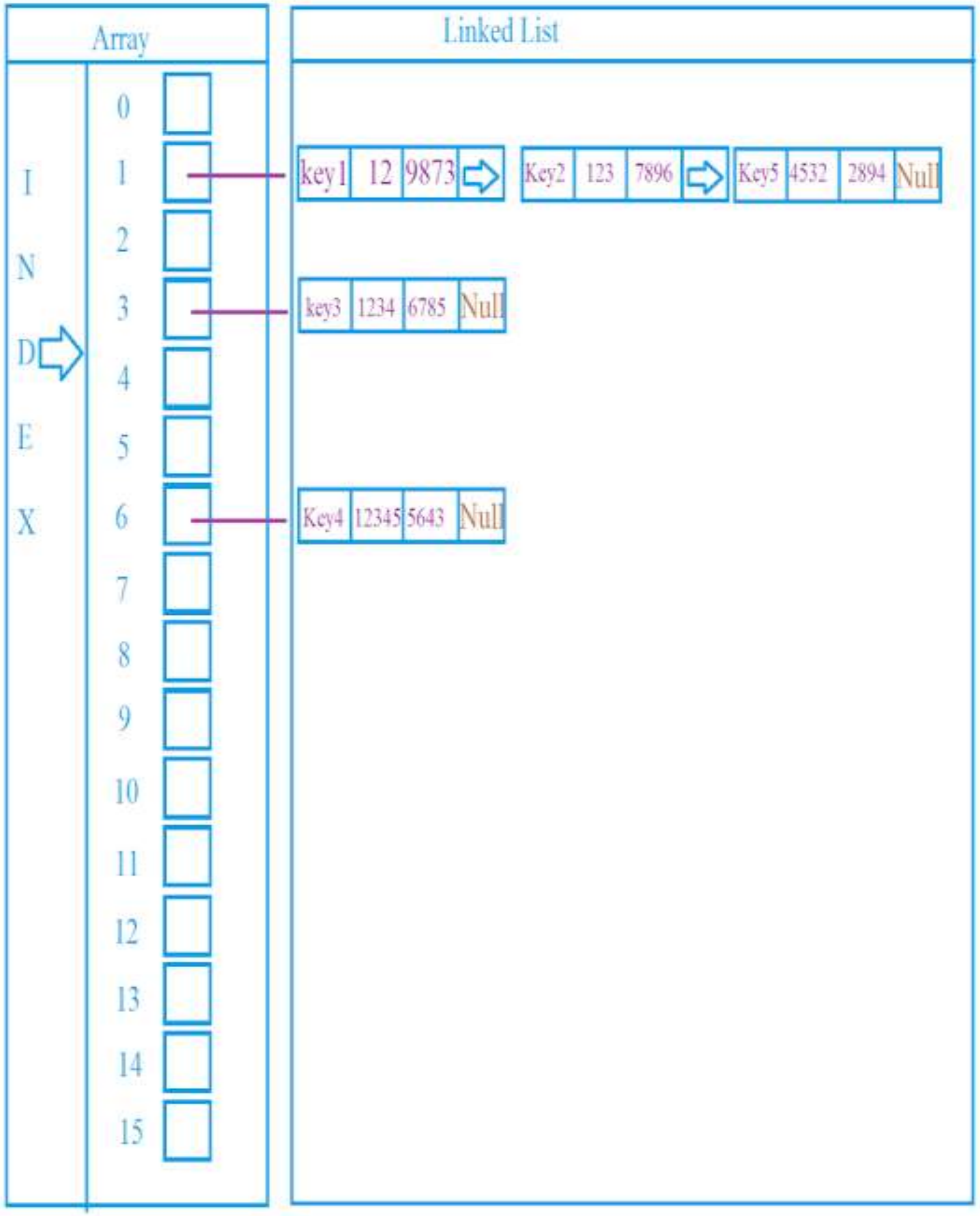


Figure: Allocation of nodes in Bucket

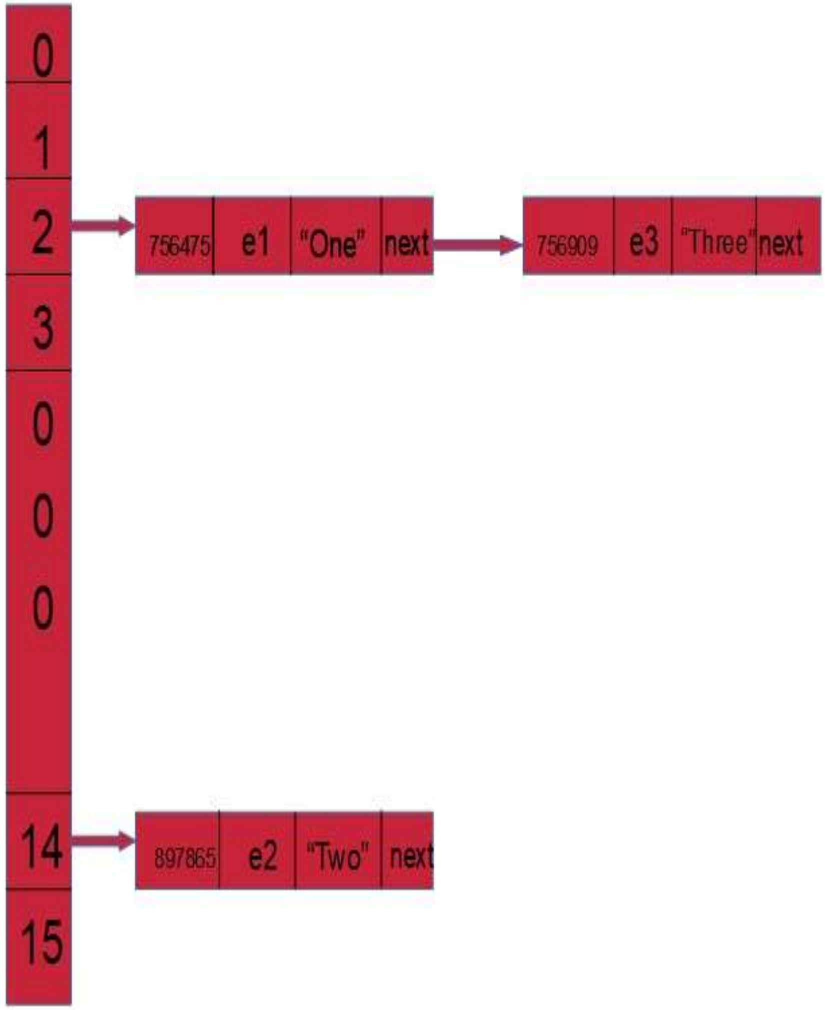
```
hashMap.put("key1",12);

 //generated hash code is 9873
 //array index evaluated from hashcode = 1

hashMap.put("key2",123);
 //generated hash code is 7896
//array index evaluated from hashcode=1


hashMap.put("key3",1234);
 //generated hash code is 6785
  //array index evaluated from hashcode=3


hashMap.put("key4",12345);
//generated hash code is 5643
 //array index evaluated from hashcode=6


hashMap.put("key5",4532);
//generated hashcode is 2894
//array index evaluated from hashcode=1
```


Array
Linked List
INDEX
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
key1 | 12 | 9873
Key2 | 123 | 7896
Key5 | 4532 | 2894 | Null
key3 | 1234 | 6785 | Null
Key4 | 12345 | 5643 | Null

How does put method work in HashMap

```
Employee e1 = new Employee("Alpha")
Employee e2 = new Employee("Beta")
Employee e3 = new Employee("Charlie")

HashMap<Employee, String> hm = new
HashMap<>()

hm.put(e1, "One");
hm.put(e2, "Two");
hm.put(e3, "Three");

Equals()
```

hashcode() for e1 – 756475 → 2
hashcode() for e2 – 897865 → 14
hashcode() for e3 – 756909 → 2



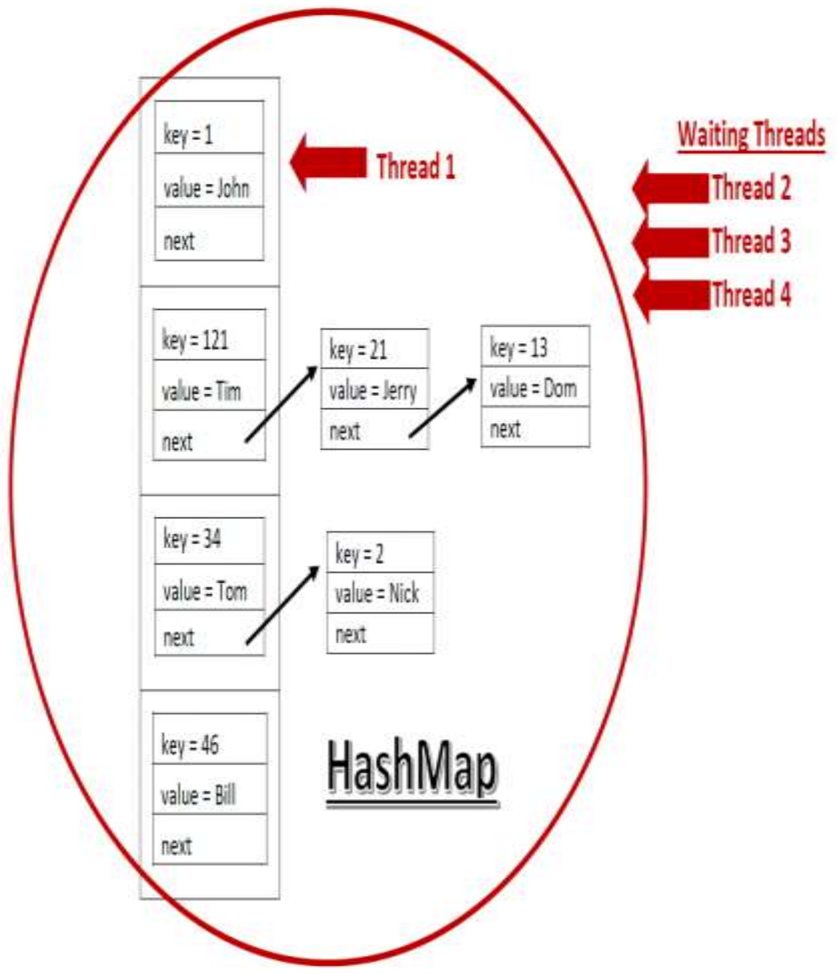
HashTable:

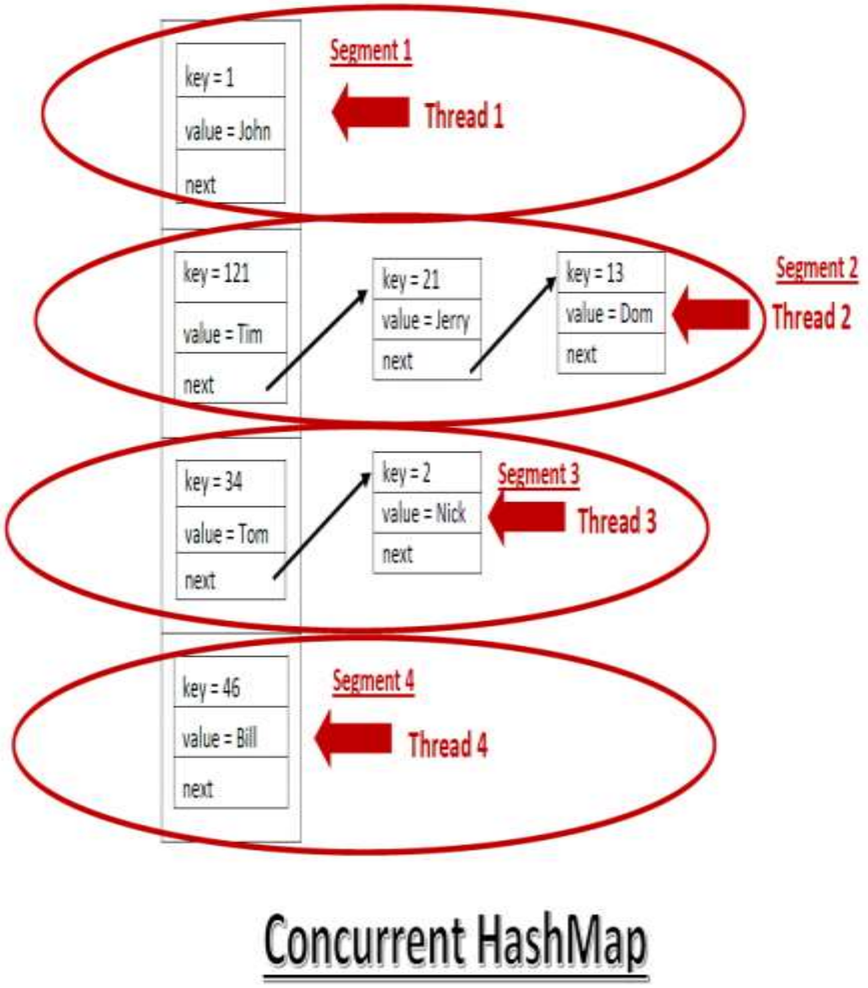
- Data Structure – HashTable
- Store Key , value
- Null key is not allowed
- Syntax :
- Map<Integer, String> map=new HashTable();
- Methods:
- put(k,v) , get(k), getKey(), containsKey(k), containsValue()
- EntrySet<key,value>

ConcurrentHashMap :

- If we try to modify the collection while iterating over it, we get ConcurrentModificationException.
- Java 1.5 introduced Concurrent classes in the java.util.concurrent package to overcome this scenario.
- ConcurrentHashMap is the Map implementation that allows us to modify the Map while iteration.
- The ConcurrentHashMap operations are thread-safe.
- ConcurrentHashMap doesn’t allow null for keys and values

	HashMap	ConcurrentHashMap
Synchronized Internally?	✗	✓
Thread Safe	✗	✓
Introduced In	JDK 1.2	JDK 1.5
Package	java.util	java.util.concurrent
Null key	✓	✗
Null Value	✓	✗
Nature Of Iterators	Fail-Fast	Fail-Safe
Fast or Slow?	Fast	Slow
Where To Use?	Single threaded Applications	Multi threaded applications


key = 1
value = John
next
Thread 1
Waiting Threads
Thread 2
Thread 3
Thread 4
key = 121
value = Tim
next
key = 21
value = Jerry
next
key = 13
value = Dom
next
key = 34
value = Tom
next
key = 2
value = Nick
next
key = 46
value = Bill
next
HashMap


Segment 1
key = 1
value = John
next
Thread 1
Segment 2
key = 121
value = Tim
next
key = 21
value = Jerry
next
key = 13
value = Dom
next
Thread 2
Segment 3
key = 34
value = Tom
next
key = 2
value = Nick
next
Thread 3
Segment 4
key = 46
value = Bill
next
Thread 4
Concurrent HashMap

➢ CopyOnWriteArrayList :

➢CopyOnWriteArrayList is a thread-safe variant of ArrayList where operations which can change the ArrayList (add, update, set methods) creates a clone of the underlying array.

➢CopyOnWriteArrayList is to be used in a Thread based environment where read operations are very frequent and update operations are rare.

➢Iterator of CopyOnWriteArrayList will never throw ConcurrentModificationException.

➢Any type of modification to CopyOnWriteArrayList will not reflect during iteration since the iterator was created.

➢List modification methods like remove, set and add are not supported in the iteration. This method will throw UnsupportedOperationException.

➢null can be added to the list.

➢ CopyOnWriteArraySet :

➢CopyOnWriteArraySet class uses CopyOnWriteArrayList internally for all of its operations and thus possesses the basic properties of CopyOnWriteArrayList.

➢CopyOnWriteArraySet is a thread-safe.

➢CopyOnWriteArraySet is to be used in Thread based environment where read operations are very frequent and update operations are rare.

➢Iterator of CopyOnWriteArraySet will never throw ConcurrentModificationException.

➢Any type of modification to CopyOnWriteArraySet will not reflect during iteration since the iterator was created.

➢Set modification methods like remove, set and add are not supported in the iteration. This method will throw UnsupportedOperationException.

- String :
- Def : Set of characters
- String as a class or String as a data type
- Syntax
- Data type :   String s;

s =“Pune”;   //Literal   (String Constant Pool)
- Non primitive data type

- Class
- String s;

s =new String(“Pune”); // JVM -> Heap