netlink
digital solutions

Collection

www.netlink.com

Why Collection ?

Arrays	Collection
Arrays are fixed in size that is once we create an array we can not increased or decreased based on our requirement.	Collection are growable in nature that is based on our requirement. We can increase or decrease of size.
With respect to memory Arrays are not recommended to use.	With respect to memory collection are recommended to use.
With respect to performance Arrays are recommended to use.	With respect to performance collection are not recommended to use.
Arrays can hold only homogeneous data types elements.	Collection can hold both homogeneous and and heterogeneous elements.
There is no underlying data structure for arrays and hence ready made method support is not available.	Every collection class is implemented based on some standard data structure and hence for every requirement ready made method support is available being a performance. we can use these method directly and We are not responsible to implement these methods.
Arrays can hold both object and primitive.	Collection can hold only object types but primitive.

What is Collection ?

What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

What is Collection framework

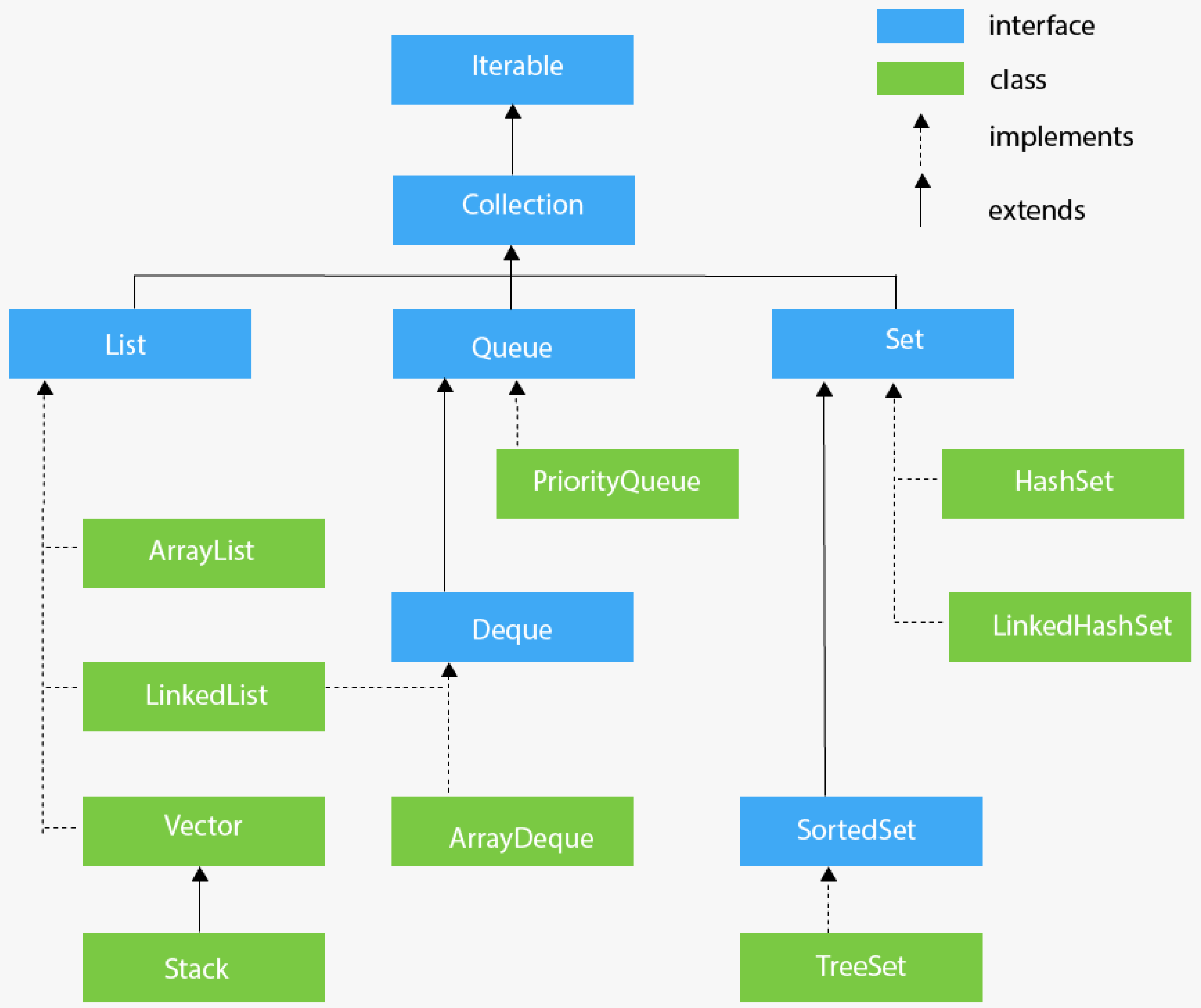
The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

- Interfaces and its implementations, i.e., classes
- Algorithm

What is Collection

- Any group of individual objects which are represented as a single unit is known as the collection of the objects.
- In Java, a separate framework named the *“**Collection Framework**”* has been defined in JDK 1.2 which holds all the collection classes and interface in it.
- The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

Collection Hierarchy



Collection interface

If we want to Represent a Group of Individual Objects as a Single Entity then we should go for Collections.

Collection Interface is considered as Root Interface of Collection Framework.

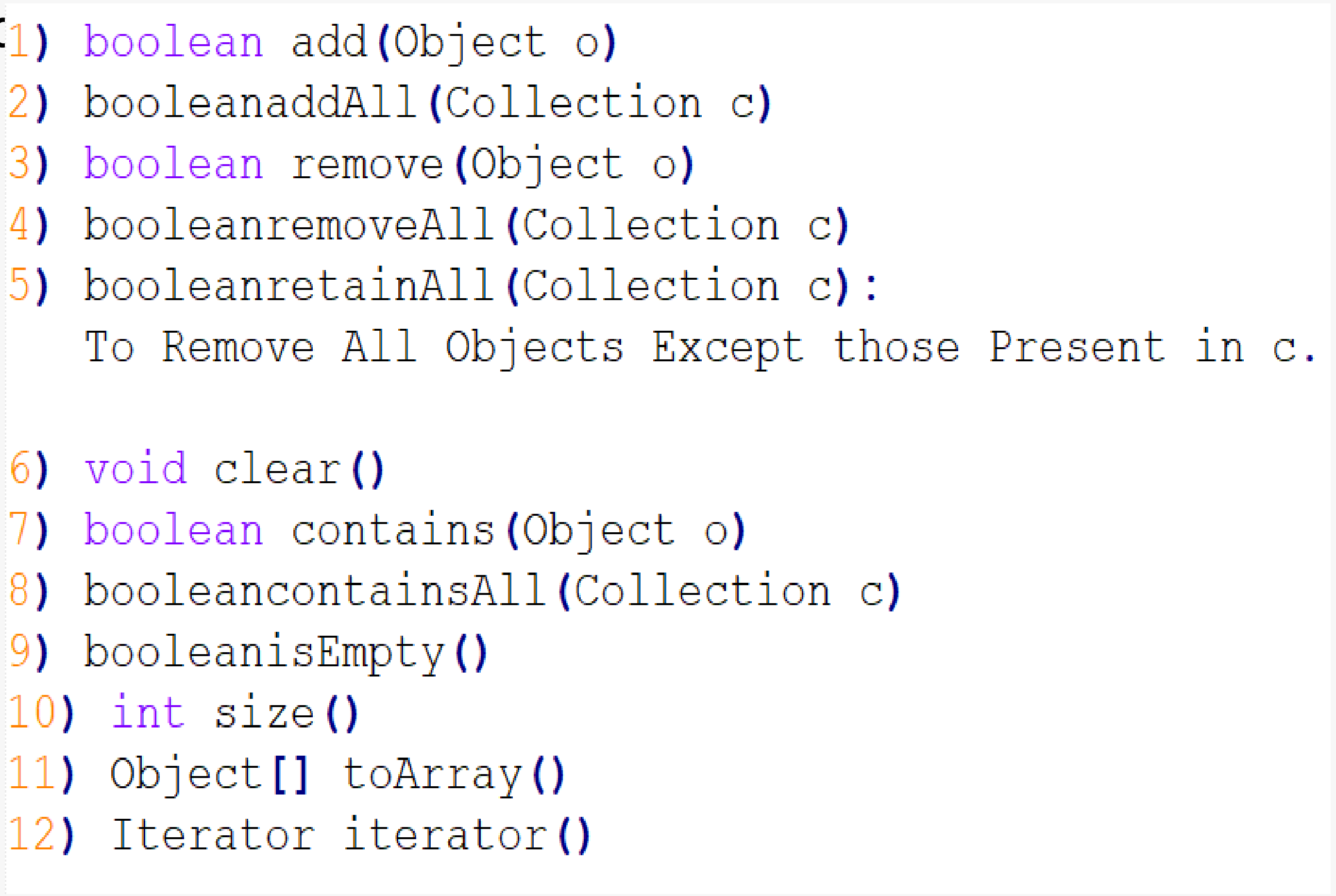
Collection Interface defines the Most Common Methods which are Applicable for any Collection Object.

Difference Between Collection (I) and Collections (C):

- Collection is an Interface which can be used to Represent a Group of Individual Objects as a Single Entity.
- Whereas Collections is an Utility Class Present in **java.util** Package to Define Several Utility Methods for Collection Objects.

Methods of Collection interface

- Collection Interface defines the Most Common Methods which are Applicable for any Collection Objects.
- The following is the List of the Methoc



```
1) boolean add(Object o)
2) booleanaddAll(Collection c)
3) boolean remove(Object o)
4) booleanremoveAll(Collection c)
5) booleanretainAll(Collection c):
   To Remove All Objects Except those Present in c.

6) void clear()
7) boolean contains(Object o)
8) booleancontainsAll(Collection c)
9) booleanisEmpty()
10) int size()
11) Object[] toArray()
12) Iterator iterator()
```

Note:

- There is No Concrete Class which implements Collection Interface Directly.
- There is No Direct Method in Collection Interface to get Objects.

Iterable interface

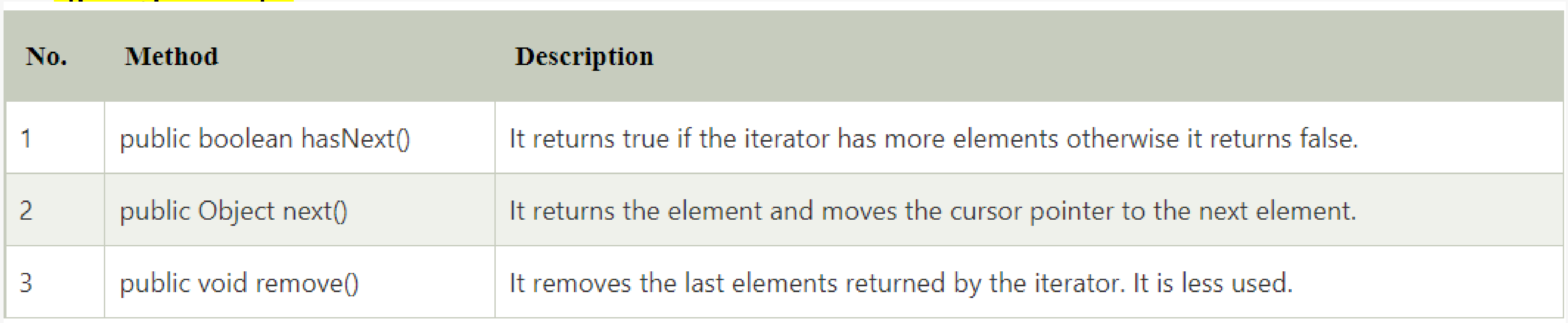
- The Iterable interface is the root interface for all the collection classes.
- The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract metho

- Iterator<T> iterator() ;

NOTE :

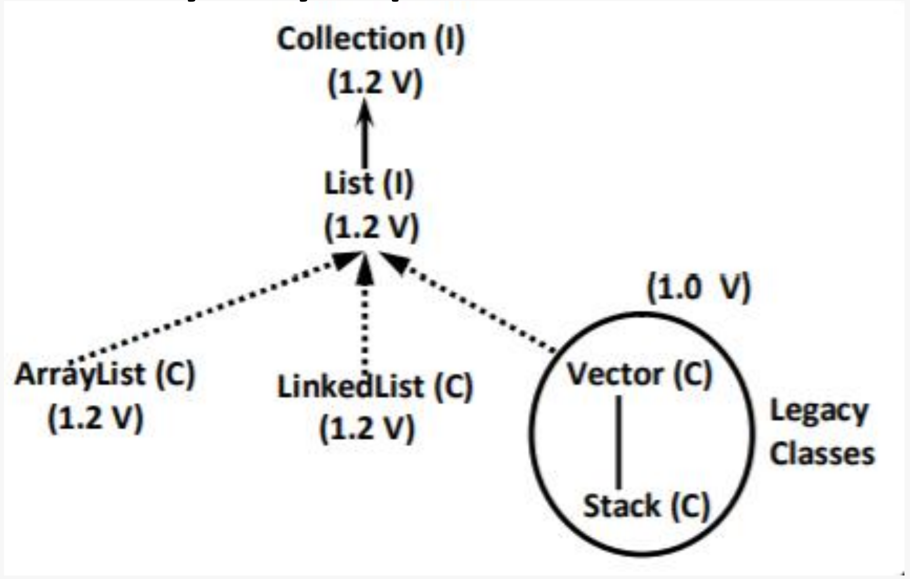
- Iterator interface provides the facility of iterating the elements in a forward



No.	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

List Interface

- It is the Child Interface of Collection.
- If we want to Represent a Group of Individual Objects where **Duplicates are allowed** and **Insertion Order Preserved**. Then we should go for List.
- We can Preserve Insertion Order and we can Differentiate
- **Duplicate Object by using Index. Hence Index will Play Very Important Role in List.**



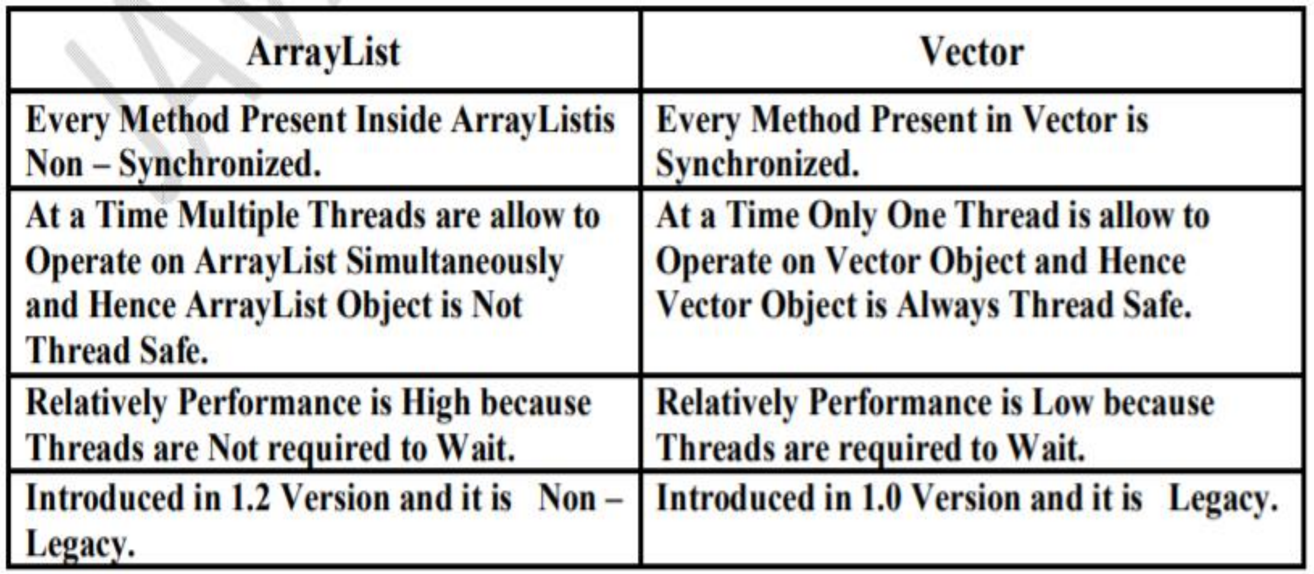
Methods: List Interface Defines the following Specific Methods

```
1) void add(int index, Object o)
2) booleanaddAll(int index, Collection c)
3) Object get(int index)
4) Object remove(int index)
5) Object set(int index, Object new):
   To Replace the Element Present at specified Index
   with provided Object and Returns Old Object.

6) intindexOf(Object o):
   Returns Index of 1st Occurrence of 'o'

7) intlastIndexOf(Object o)
8) ListIteratorlistIterator();
```

ArrayList

The Underlying Data Structure for ArrayList is Resizable Array OR Growable Array.

- Duplicate Objects are allowed.
- Insertion Order is Preserved.
- Heterogeneous Objects are allowed (**Except** TreeSet and TreeMap Everywhere Heterogeneous Objects are allowed).
- Null Insertion is Possible.



ArrayList	Vector
Every Method Present Inside ArrayListis Non – Synchronized.	Every Method Present in Vector is Synchronized.
At a Time Multiple Threads are allow to Operate on ArrayList Simultaneously and Hence ArrayList Object is Not Thread Safe.	At a Time Only One Thread is allow to Operate on Vector Object and Hence Vector Object is Always Thread Safe.
Relatively Performance is High because Threads are Not required to Wait.	Relatively Performance is Low because Threads are required to Wait.
Introduced in 1.2 Version and it is Non – Legacy.	Introduced in 1.0 Version and it is Legacy.

Constructors :

1) ArrayList l = new ArrayList();

- Creates an Empty ArrayList Object with Default Initial Capacity 10.
- If ArrayList Reaches its Max Capacity then a New ArrayList Object will be Created with

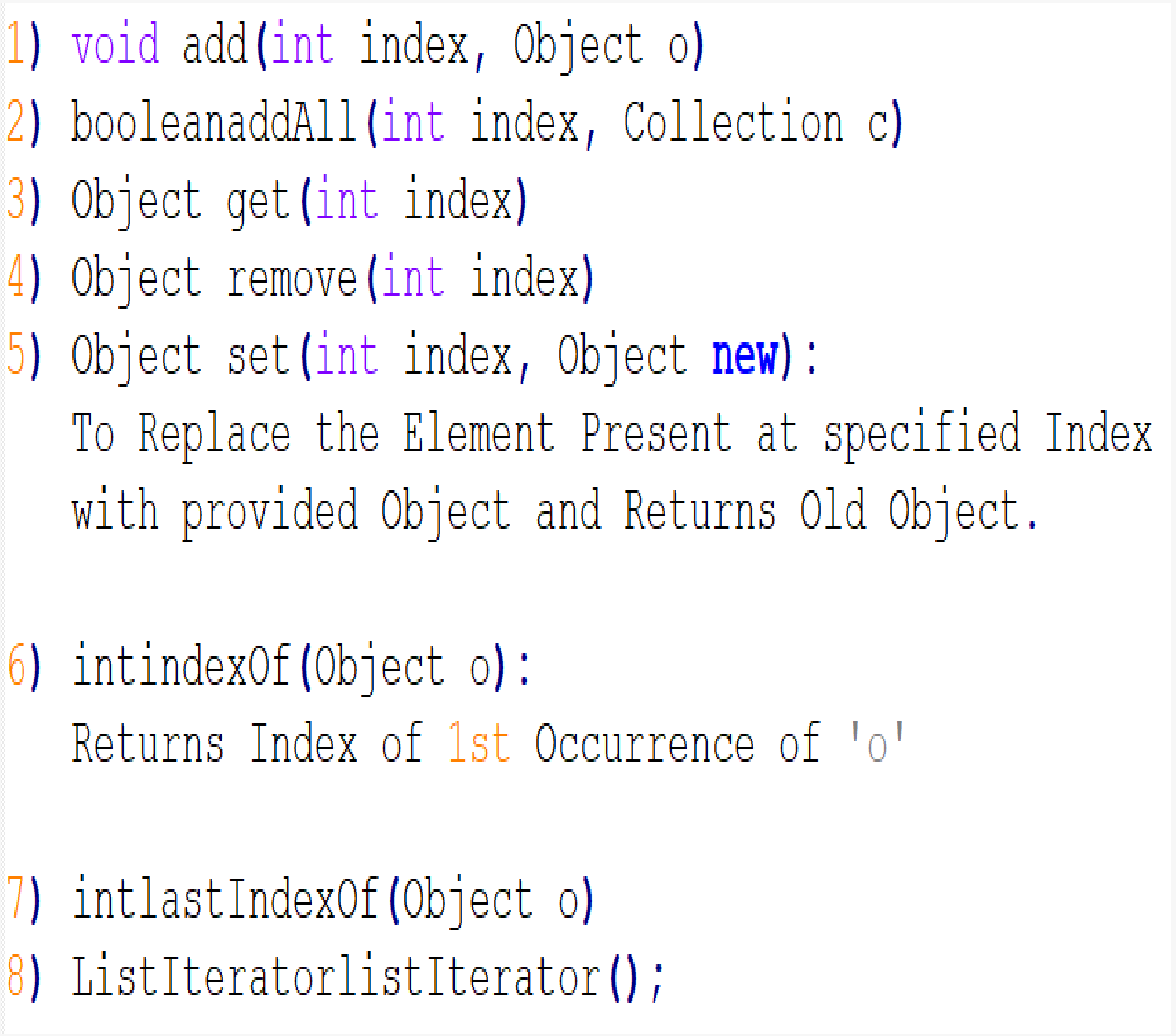
2) ArrayList l = new ArrayList(intinitialCapacity);

- Creates an Empty ArrayList Object with specified Initial Capacity

3) ArrayList l = new ArrayList(Collection c);

- Creates an EqualentArrayList Object for the given Collection Object.
- This Constructor Meant for Inter Conversion between Collection Objects.

ArrayList Methods



```
1) void add(int index, Object o)
2) booleanaddAll(int index, Collection c)
3) Object get(int index)
4) Object remove(int index)
5) Object set(int index, Object new):
   To Replace the Element Present at specified Index
   with provided Object and Returns Old Object.

6) intindexOf(Object o):
   Returns Index of 1st Occurrence of 'o'

7) intlastIndexOf(Object o)
8) ListIteratorlistIterator();
```

LinkedList

- The Underlying Data Structure is **Double LinkedList.**
- Insertion Order is Preserved.
- Duplicate Objects are allowed.
- Heterogeneous Objects are allowed.
- null Insertion is Possible.
- Implements Serializable and Cloneable Interfaces but Not RandomAccessInterface.
- Best Choice if Our Frequent Operation is **InsertionOR Deletion in the Middle.**
- Worst Choice if Our Frequent Operation is Retrieval

Constructors :

1) LinkedList l = new LinkedList();

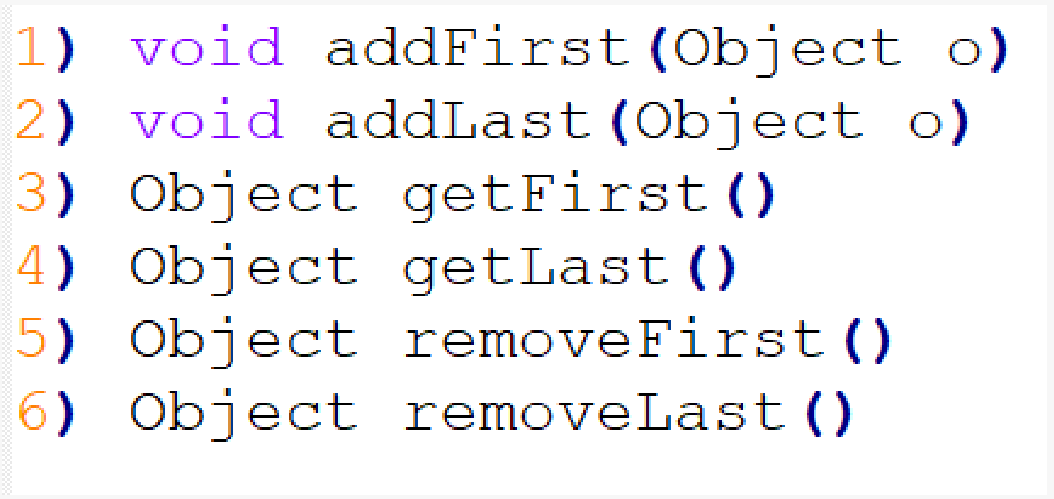
Creates an Empty LinkedList Object.

2) LinkedList l = new LinkedList(Collection c);

Creates an Equivalent LinkedList Object for the given Collection

LinkedList Methods

Methods:
Usually we can Use LinkedList to Implement Stacks and Queues.
To Provide Support for this Requirement LinkedList Class Defines the following 6 Specific Methods.



```
1) void addFirst(Object o)
2) void addLast(Object o)
3) Object getFirst()
4) Object getLast()
5) Object removeFirst()
6) Object removeLast()
```

Vector

The Underlying Data Structure is Resizable Array OR Growable Array.

Insertion Order is Preserved.

Duplicate Objects are allowed.

Heterogeneous Objects are allowed.

null Insertion is Possible.

Implements Serializable, Cloneable and RandomAccess interfaces.

Every Method Present Inside Vector is Synchronized and Hence Vector Object is Thread Safe.

Vector is the Best Choice if Our Frequent Operation is Retrieval.

Worst Choice if Our Frequent Operation is Insertion OR Deletion in the Middle.

Constructors :

1) **Vector v = new Vector()**;

- Creates an Empty Vector Object with Default Initial Capacity 10.
- Once Vector Reaches its Max Capacity then a New Vector Object will be Created with

2) **Vector v = new Vector(intinitialCapacity)**;

3) **Vector v = new Vector(intinitialCapacity, intincrementalCapacity);**

4) **Vector v = new Vector(Collection c);**

Vector Methods

```
1) To Add Elements:
   ☐ add(Object o)-> Collection
   ☐ add(int index, Object o) ->List
   ☐ addElement(Object o) -> Vector

2) To Remove Elements:
   ☐ remove(Object o) -> Collection
   ☐ removeElement(Object o)-> Vector
   ☐ remove(int index) -> List
   ☐ removeElementAt(int index)-> Vector
   ☐ clear() -> Collection
   ☐ removeAllElements() -> Vector

3) To Retrive Elements:
   ☐ Object get(int index)-> List
   ☐ Object elementAt(int index)-> Vector
   ☐ Object firstElement() -> Vector
   ☐ Object lastElement()->Vector

4) Some Other Methods:
   ☐ int size()
   ☐ int capacity()
   ☐ Enumeration element()
```

Stack

- ❑ It is the Child Class of Vector.
- ❑ It is a Specially Designed Class for Last In First Out (LIFO) Order.
- ❑ We've seen that the Stack class is a subclass of java.util.Vector. The Vector class is synchronized. It uses the traditional way to achieve thread-safety: making its methods "synchronized." As its subclass, the Stack class is **synchronized** as well.

Method Of Stack

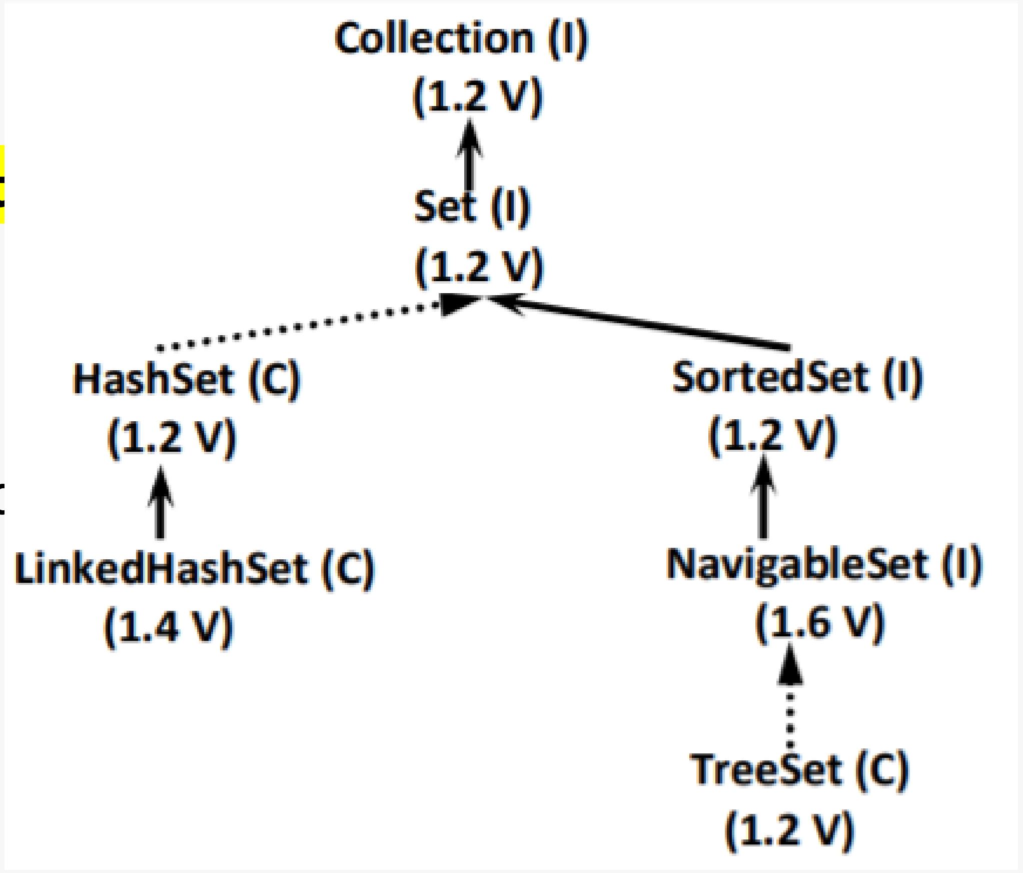
```
1) Object push(Object o);
   To Insert an Object into the Stack.

2) Object pop();
   To Remove and Return Top of the Stack.

3) Object peek();
   Ro Return Top of the Stack without Removal.

4) boolean empty();
   Returns true if Stack is Empty

5) int search(Object o);
   Returns Offset if the Element is Available Otherwise Returns -1
```

Set Interface

- It is the Child Interface of Collection.

- If we want to Represent a Group of Individual Objects as a Single Entity where **Duplicates are Not allowed, and Insertion Order is Not Preserved**, then we should go for Set.

- Set Interface doesn't contain any New Methods anc Hence we have to Use Only Collection Interface Methods



HashSet

- The Underlying Data Structure is Hashtable.
- Insertion Order is Not Preserved and it is Based on hashCode of the Objects.
- Duplicate Objects are Not Allowed.
- If we are trying to Insert Duplicate Objects then we won't get any Compile Time OR Runtime Error.add() Simply Returns false.
- null Insertion is Possible.
- Heterogeneous objects are allowed.
- HashSet implements Serializable and Cloneable Interfaces but Not RandomAccess.
- **If Our Frequent Operation is Search Operation,** then HashSet is the Best Choice.

Constructors:

1) HashSet h = new HashSet();
Creates an Empty HashSet Object with Default Initial Capacity ***16*** and Default Fill Ratio : 0.75.

2) HashSet h = new HashSet(intinitialCapacity);
Creates an Empty HashSet Object with specified Initial Capacity and Default Fill Ratio : 0.75.

3) HashSet h = new HashSet(intinitialCapacity, float fillRatio);

4) HashSet h = new HashSet(Collection c);

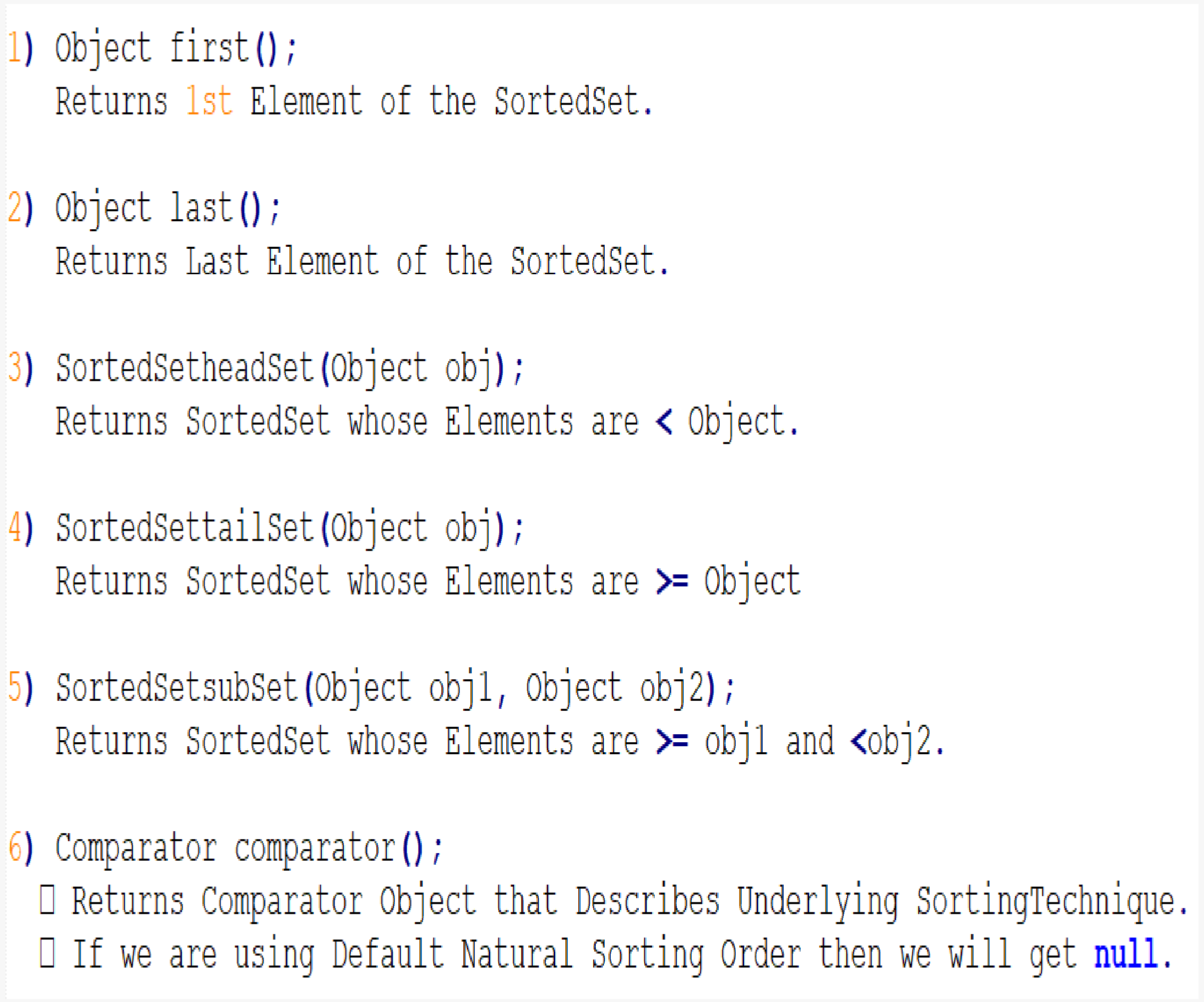
LinkedHash Set

HashSet	LinkedHashSet
The Underlying Data Structure is Hashtable.	The Underlying Data Structure is a Combination of **LinkedList and Hashtable.**
Insertion Order is Not Preserved.	Insertion Order is Preserved.
Introduced in 1.2 Version.	Introduced in 1.4 Version

SortedSet:

- It is the Child Interface of Set.
- If we want to Represent a Group of Individual Objects without Duplicates and all Objects will be Inserted According to **Some Sorting Order,** then we should go for SortedSet.
- The Sorting can be Either Default Natural Sorting OR Customized Sorting Order.
- For String Objects Default Natural Sorting is Alphabetical Order.
- For Numbers Default Natural Sorting is Ascending Order.

Methods Of Sorted Set



```
1) Object first();
   Returns 1st Element of the SortedSet.

2) Object last();
   Returns Last Element of the SortedSet.

3) SortedSetheadSet(Object obj);
   Returns SortedSet whose Elements are < Object.

4) SortedSettailSet(Object obj);
   Returns SortedSet whose Elements are >= Object

5) SortedSetsubSet(Object obj1, Object obj2);
   Returns SortedSet whose Elements are >= obj1 and <obj2.

6) Comparator comparator();
  □ Returns Comparator Object that Describes Underlying SortingTechnique.
  □ If we are using Default Natural Sorting Order then we will get null.
```

TreeSet:

- The Underlying Data Structure is **Balanced Tree.**
- Insertion Order is Not Preserved and it is Based on Some Sorting Order.
- **Heterogeneous Objects are Not Allowed.** If we are trying to Insert we will get Runtime Exception Saying ClassCastException.
- Duplicate Objects are Not allowed.
- null Insertion is Possible (Only Once).
- Implements Serializable and Cloneable Interfaces but Not RandomAccess Interface.

null Acceptance:

- From 1.7 onwards null is not at all accepted by TreeSet.

Constructors:

1) TreeSet t = new TreeSet();

Creates an Empty TreeSet Object where all Elements will be Inserted According to Default Natural Sorting Order.

2) TreeSet t = new TreeSet(Comparator c);

Creates an Empty TreeSet Object where all Elements will be Inserted According to Customized Sorting Order which is described by Comparator Object.

3) TreeSet t = new TreeSet(Collection c);

4) TreeSet t = new TreeSet(SortedSet s);

Note:

 If we are Depending on Default Natural Sorting Order Compulsory Objects should be Homogeneous and Comparable. Otherwise we will get RE: ClassCastException.

 An object is said to be Comparable if and only if corresponding class implements Comparable interface.

 All Wrapper Classes, String Class Already Implements Comparable Interface. But StringBuffer Class doesn't Implement Comparable Interface.

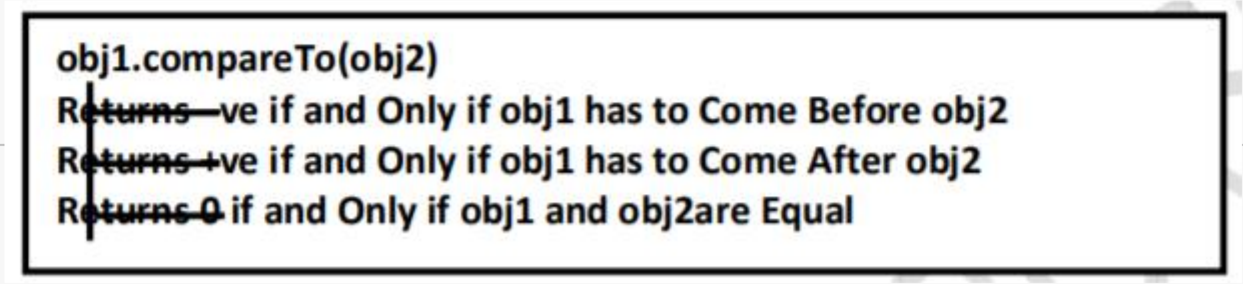
Comparable (I):

- Comparable Interface Present in java.lang Package
- It contains Only One Method **compareTo().**
- Method:

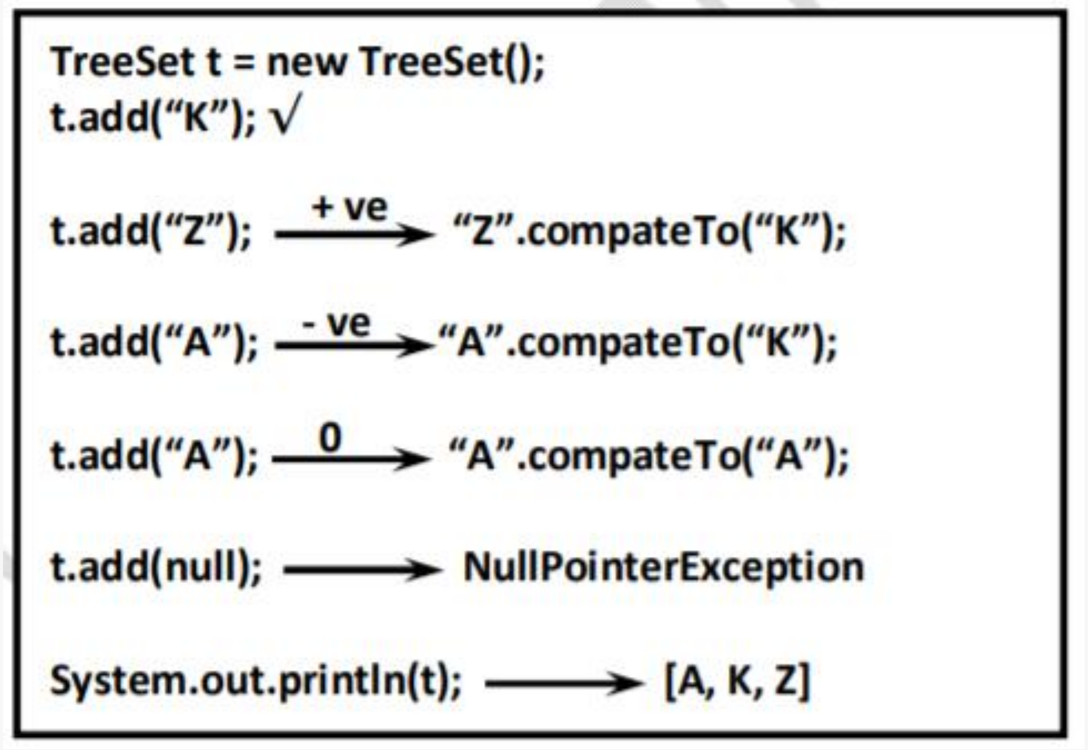
public int compareTo(Object o);

NOTE :
Wheneverwe are Depending on Default Natural Sorting Order and if we are trying to Insert Elements then Internally JVM will Call compareTo() to IdentifySorting Order.

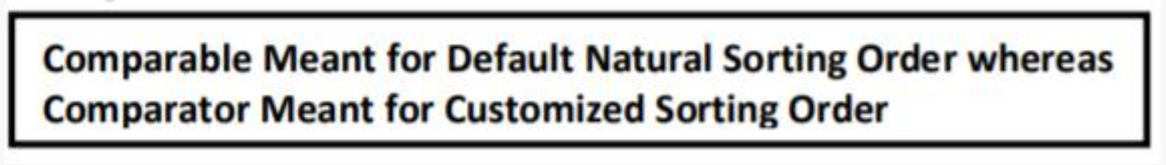
NOTE :
If we are Not satisfied with Default Natural Sorting Order OR if Default Natural Sorting Order is Not Already Available then we can Define Our Own Sorting by using **Comparator Object**



obj1.compareTo(obj2)
Returns –ve if and Only if obj1 has to Come Before obj2
Returns +ve if and Only if obj1 has to Come After obj2
Returns 0 if and Only if obj1 and obj2are Equal



TreeSet t = new TreeSet();
t.add("K"); √
t.add("Z"); $\xrightarrow{+ve}$ **"Z".compateTo("K");**
t.add("A"); $\xrightarrow{-ve}$ **"A".compateTo("K");**
t.add("A"); $\xrightarrow{0}$ **"A".compateTo("A");**
t.add(null); $\longrightarrow$ **NullPointerException**
System.out.println(t); $\longrightarrow$ **[A, K, Z]**



Comparable Meant for Default Natural Sorting Order whereas Comparator Meant for Customized Sorting Order

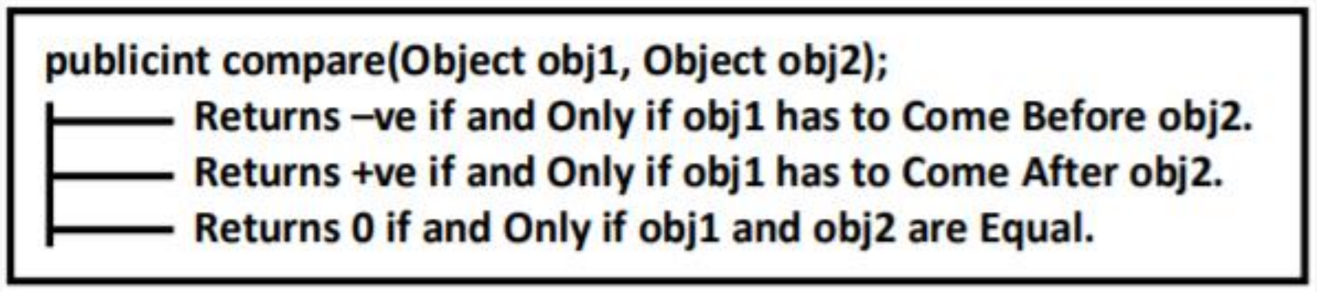
Comparator (I): // customize

➢ **This Interface Present in java.util Package.**

Methods:

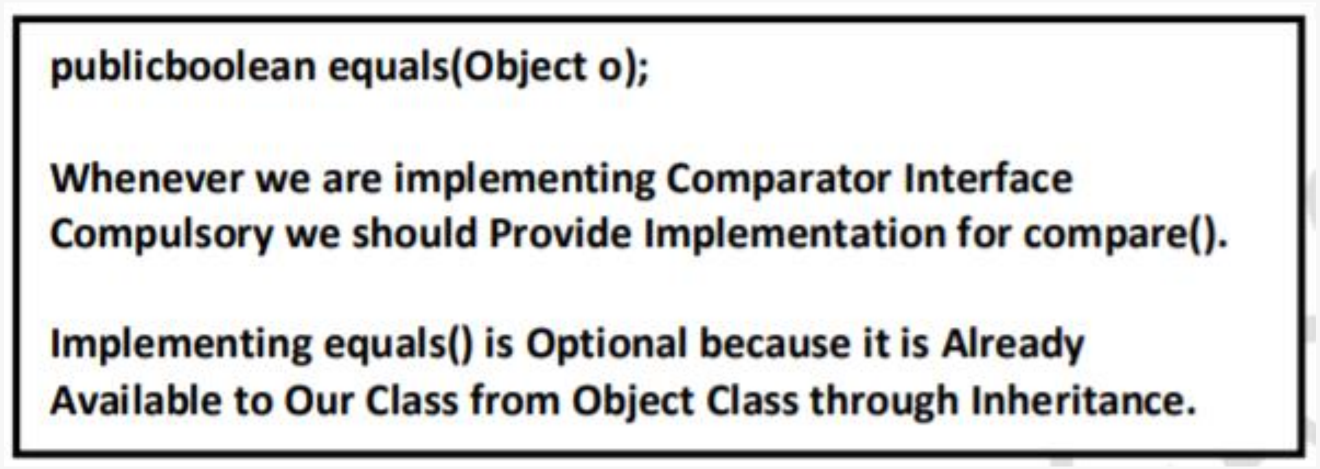
It contains 2 Methods

- compare()
- equals()



publicint compare(Object obj1, Object obj2);

- Returns –ve if and Only if obj1 has to Come Before obj2.
- Returns +ve if and Only if obj1 has to Come After obj2.
- Returns 0 if and Only if obj1 and obj2 are Equal.



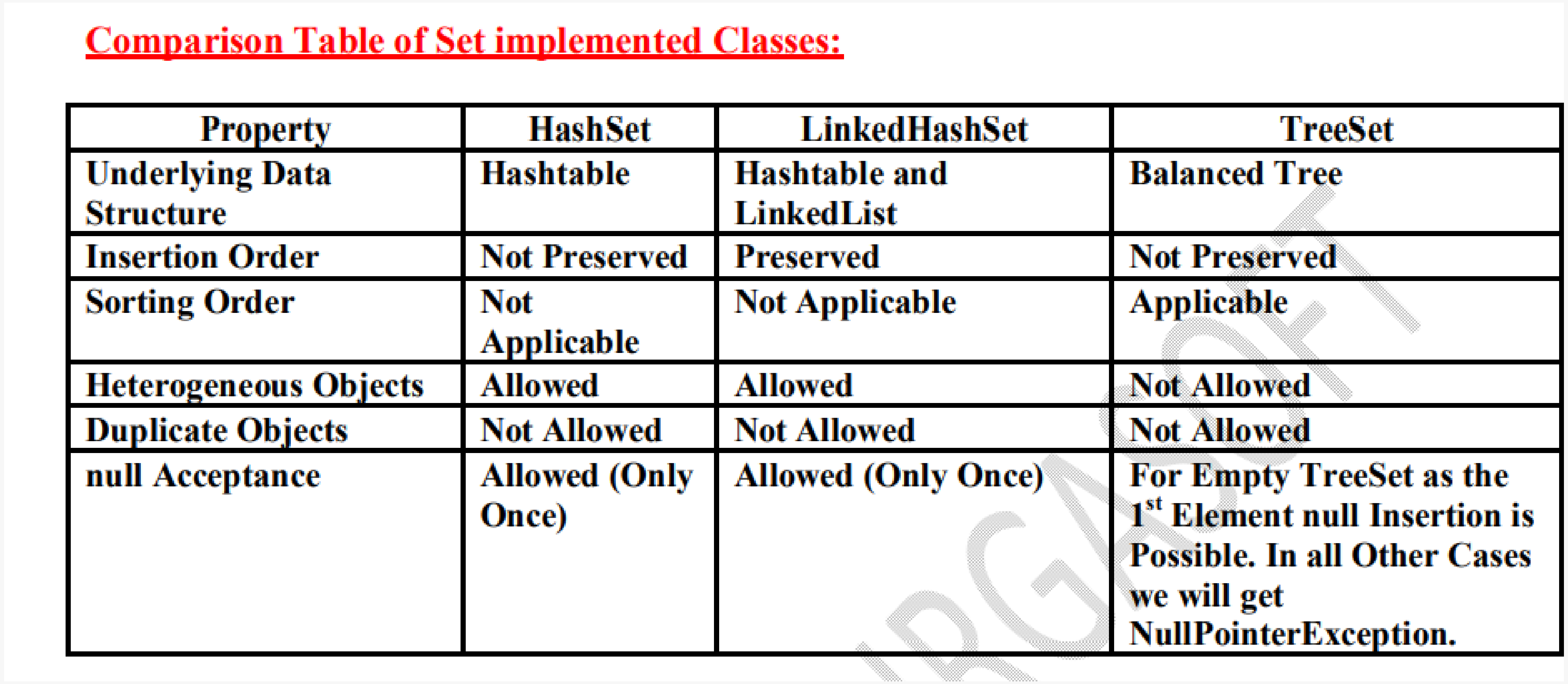
publicboolean equals(Object o);

Whenever we are implementing Comparator Interface Compulsory we should Provide Implementation for compare().

Implementing equals() is Optional because it is Already Available to Our Class from Object Class through Inheritance.

When we go for Comparable and When we go for Comparator: Comparable Vs Comparator:

- For Predefined Comparable Classes (Like String) Default Natural Sorting Order is Already Available. If we are Not satisfied with that we can Define Our Own Sorting by Comparator Object.
- For Predefine Non- Comparable Classes (Like StringBuffer) Default Natural Sorting Order is Not Already Available. If we want to Define Our Own Sorting we can Use Comparator Object.
- For Our Own Classes (Like Employee) the Person who is writing Employee Class he is Responsible to Define Default Natural Sorting Order by implementing Comparable Interface.
- The Person who is using Our Own Class if he is Not satisfied with Default Natural Sorting Order he can Define his Own Sorting by using Comparator Object.
- If he is satisfied with Default Natural Sorting Order then he can Use Directly Our Class.

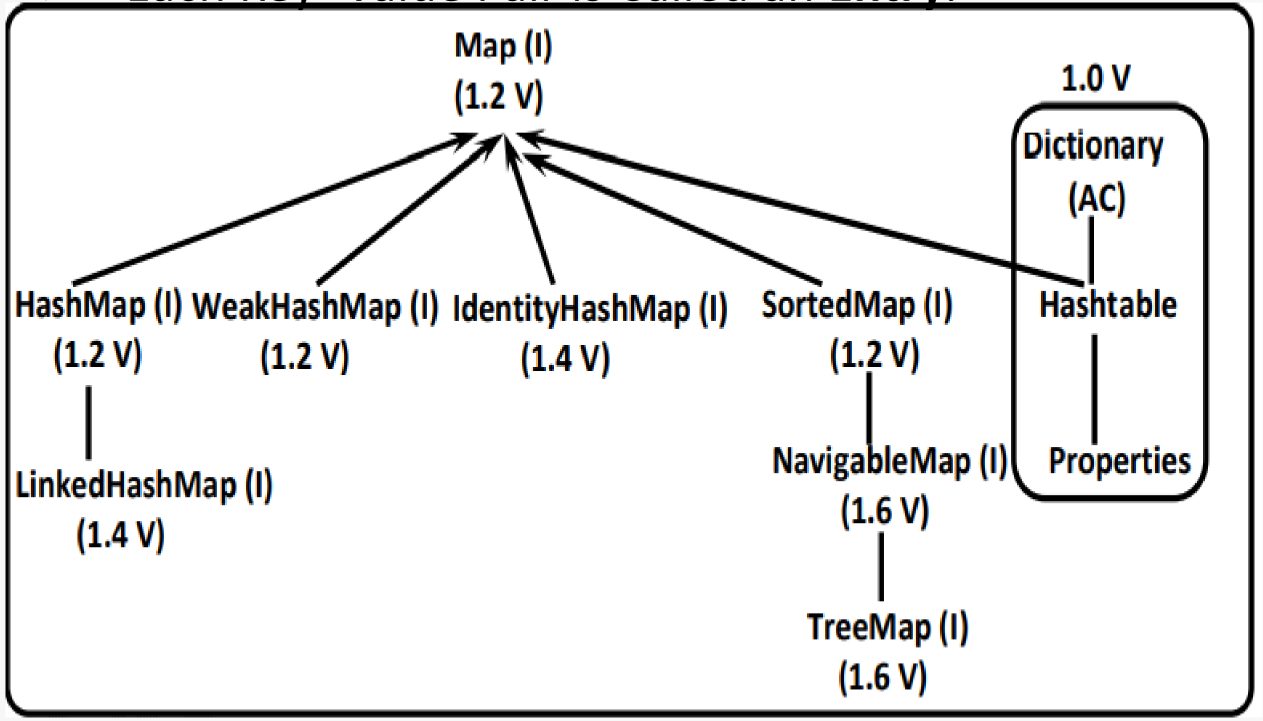


Comparison Table of Set implemented Classes:

Property	HashSet	LinkedHashSet	TreeSet
Underlying Data Structure	Hashtable	Hashtable and LinkedList	Balanced Tree
Insertion Order	Not Preserved	Preserved	Not Preserved
Sorting Order	Not Applicable	Not Applicable	Applicable
Heterogeneous Objects	Allowed	Allowed	Not Allowed
Duplicate Objects	Not Allowed	Not Allowed	Not Allowed
null Acceptance	Allowed (Only Once)	Allowed (Only Once)	For Empty TreeSet as the 1st Element null Insertion is Possible. In all Other Cases we will get NullPointerException.

Map

- Map is Not Child Interface of Collection.
- If we want to Represent a Group of Objects as Key- Value Pairs then we should go for Map.
- Both Keys and Values are Objects Only.
- **Duplicate Keys are Not allowed.** But Values can be Duplicated.
- Each Key- Value Pair is Called an **Entry**.



Methods Of Map ;

```
1) Object put(Object key, Object value);
To Add One Key- Value Pair.
If the specified Key is Already Available then Old Value will
be Replaced with New Value and Returns Old Value.

2) void putAll(Map m)
3) Object get(Object key)
4) Object remove(Object key)
5) booleancontainsKey(Object key)
6) booleancontainsValue(Object value)
7) booleanisEmpty()
8) int size()
9) void clear()
10) Set keySet()
11) Collection values()
12) Set entrySet()
```

HashMap:

- The Underlying Data Structure is Hashtable.
- Duplicate Keys are Not Allowed. But Values can be Duplicated.
- Heterogeneous Objects are allowed for Both Keys and Values.
- 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.**

Constructors:

1) HashMap m = new HashMap();
Creates an Empty HashMap Object with **Default Initial Capacity 16** and Default Fill Ratio 0.75

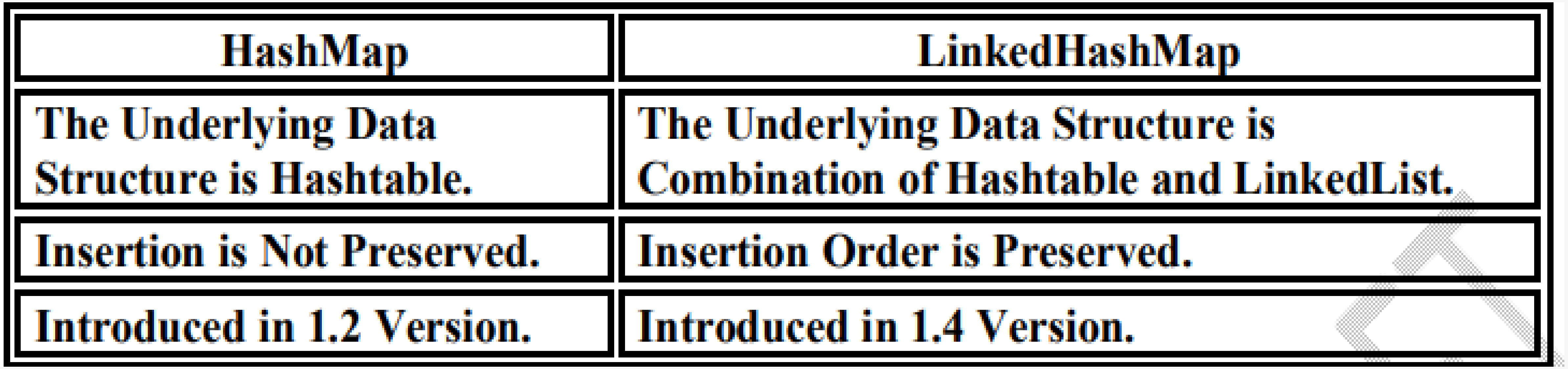
2) HashMap m = new HashMap(intinitialcapacity);

3) HashMap m = new HashMap(intinitialcapacity, float fillRatio);

4) HashMap m = new HashMap(Map m)

LinkedHashMap:

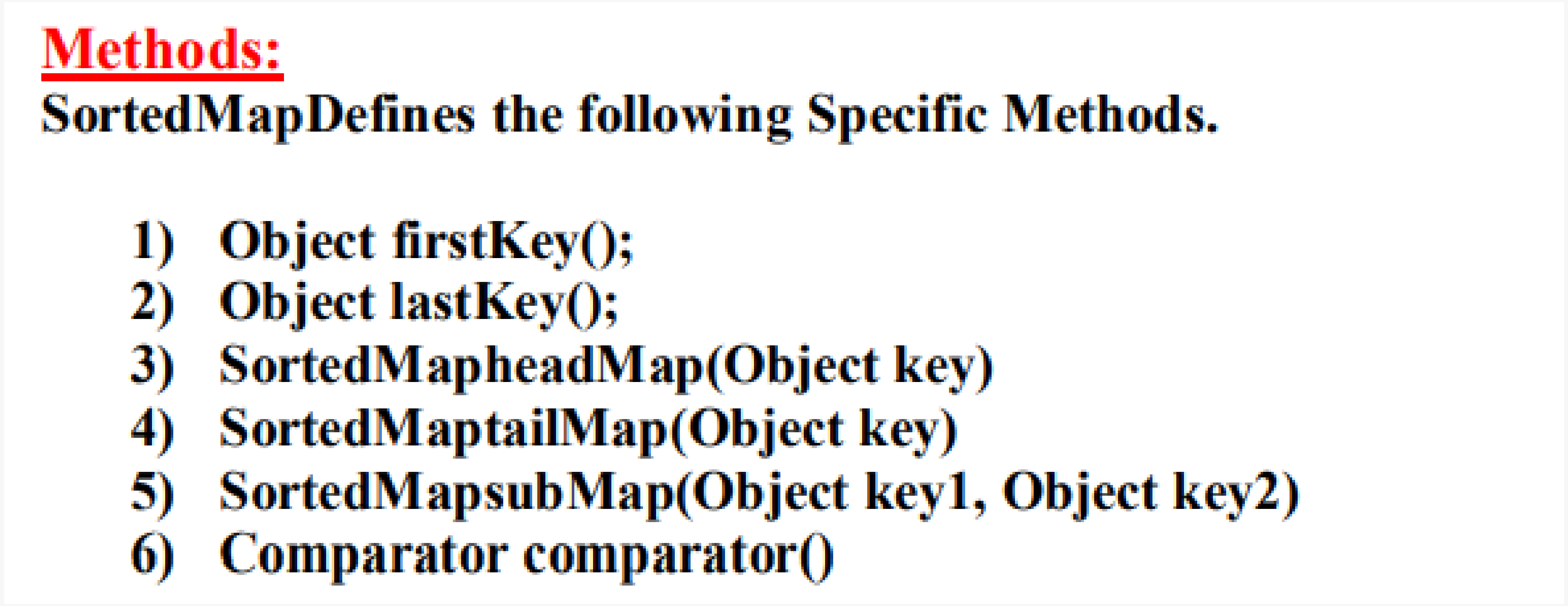
- It is the Child Class of HashMap.
- It is Exactly Same as HashMap Except the following Differeces.



HashMap	LinkedHashMap
The Underlying Data Structure is Hashtable.	The Underlying Data Structure is Combination of Hashtable and LinkedList.
Insertion is Not Preserved.	Insertion Order is Preserved.
Introduced in 1.2 Version.	Introduced in 1.4 Version.

SortedMap:

- It is the Child Interface of Map.
- If we want to Represent a Group of Key - Value Pairs According Some Sorting Order of Keys then we should go for SortedMap.



Methods:

SortedMapDefines the following Specific Methods.

1) Object firstKey();
2) Object lastKey();
3) SortedMapheadMap(Object key)
4) SortedMaptailMap(Object key)
5) SortedMapsubMap(Object key1, Object key2)
6) Comparator comparator()

TreeMap:

- The Underlying Data Structure is **Red -Black Tree.**
- Duplicate Keys are Not Allowed. But Values can be Duplicated.
- Insertion Order is Not Preserved, and it is **Based on Some Sorting Order of Keys**.
- **If we are depending on Default Natural Sorting Order, then the Keys should be Homogeneous and Comparable.** Otherwise, we will get Runtime Exception Saying ClassCastException.
- **If we defining Our Own Sorting by Comparator, then Keys can be** Heterogeneous **and NonComparable. But there are No Restrictions on Values.** They can be Heterogeneous and Non- Comparable

null Acceptance:

- For Empty TreeMap as the 1 st Entry with null Key is Allowed. But After inserting that Entry if we are trying to Insert any Other Entry we will get RE: NullPointerException.
- For Non- Empty TreeMap if we are trying to Insert null Entry then we will get Runtime Exception Saying NullPointerException.
- There are No Restrictions on null Values

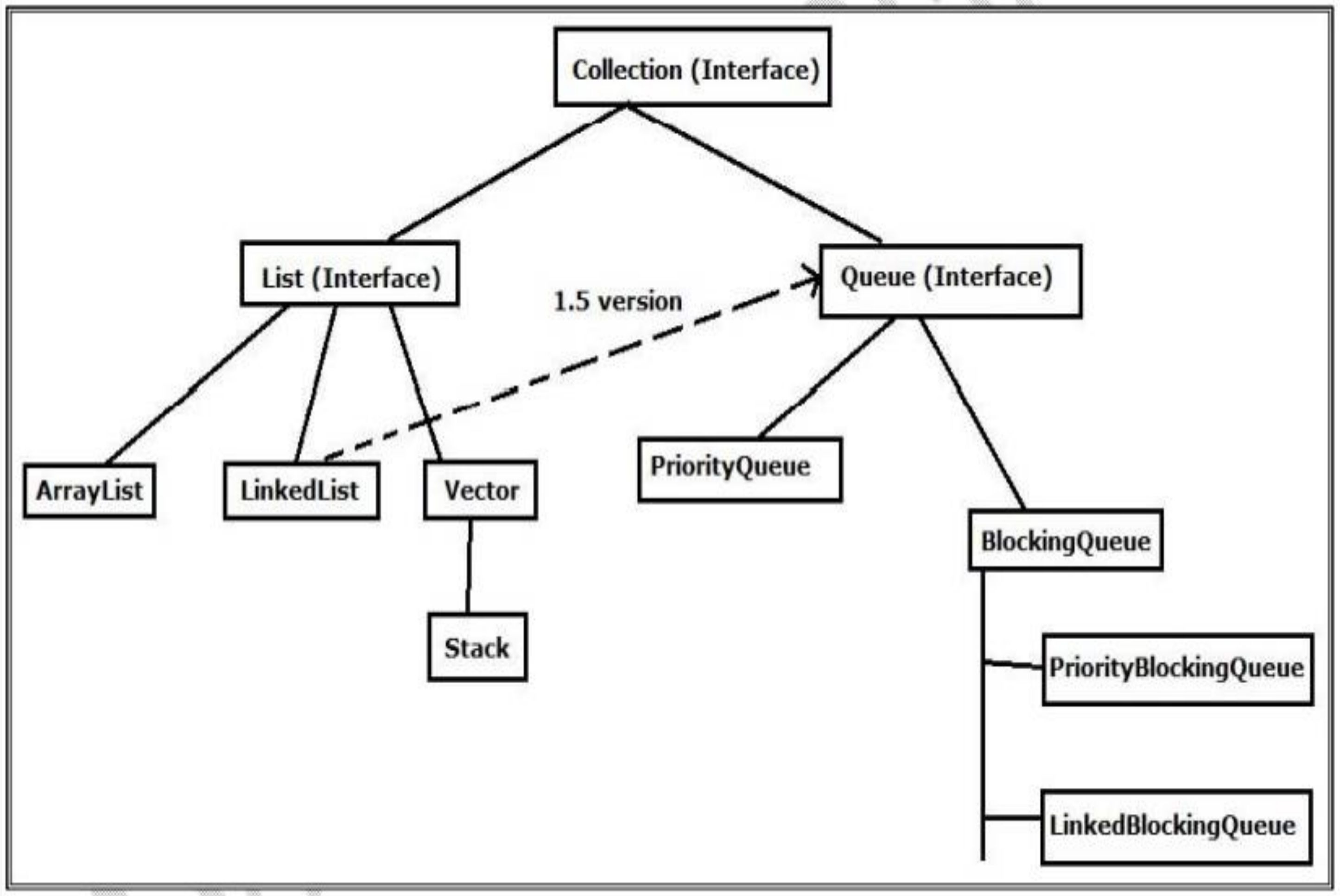
TreeMap:

Constructors:

- 1) TreeMap t = new TreeMap(); For Default Natural Sorting Order.
- 2) TreeMap t = new TreeMap(Comparator c); For Customized Sorting Order.
- 3) TreeMap t = new TreeMap(SortedMap m); Inter Conversion between Map Objects.
- 4) TreeMap t = new TreeMap(Map m);

Queue Interface

- Queue is a Child Interface of Collection.
- If we want to Represent a Group of Individual Objects Prior to processing then we should go for Queue.
- From 1.5 Version onwards LinkedList also implements Queue Interface.
- Usually Queue follows FIFO Order. But Based on Our Requirement we can Implement Our Own Priorities Also (PriorityQueue)
- **LinkedList based**



Queue Interface

Methods Of Queue Interface

Eg:

Before sending a Mail we have to Store all Mail IDs in Some Data Structure and for the 1 st Inserted Mail ID Mail should be Sent 1 st .For this Requirement Queue is the Best Choice.

```
1) boolean offer(Object o); To Add an Object into the Queue.

2) Object peek();
□ To Return Head Element of the Queue.
□ If Queue is Empty then this Method Returns null.

3) Object element();
□ To Return Head Element of the Queue.
□ If Queue is Empty then this Methodraises RE: NoSuchElementException

4) Object poll();
□ To Remove and Return Head Element of the Queue.
□ If Queue is Empty then this Method Returns null.

5) Object remove();
□ To Remove and Return Head Element of the Queue.
□ If Queue is Empty then this Method raise RE: NoSuchElementException
```

PriorityQueue:

- This is a Data Structure which can be used to Represent a Group of Individual Objects Prior to processing according to Some Priority.
- The Priority Order can be Either Default Natural Sorting Order OR Customized Sorting Order specified by Comparator Object.
- If we are Depending on Natural Sorting Order, then the Objects should be **Homogeneous** and Comparable otherwise, we will get ClassCastException.
- If we are defining Our Own Sorting by Comparator, then the Objects Need Not homogeneous and Comparable.
- Duplicate objects are Not Allowed.
- Insertion Order is Not Preserved, and it is Based on Some Priority.
- null Insertion is Not Possible Even as 1 st Element Also

Note: Some Operating Systems won't Provide Proper Support for PriorityQueues.

THANK
YOU!