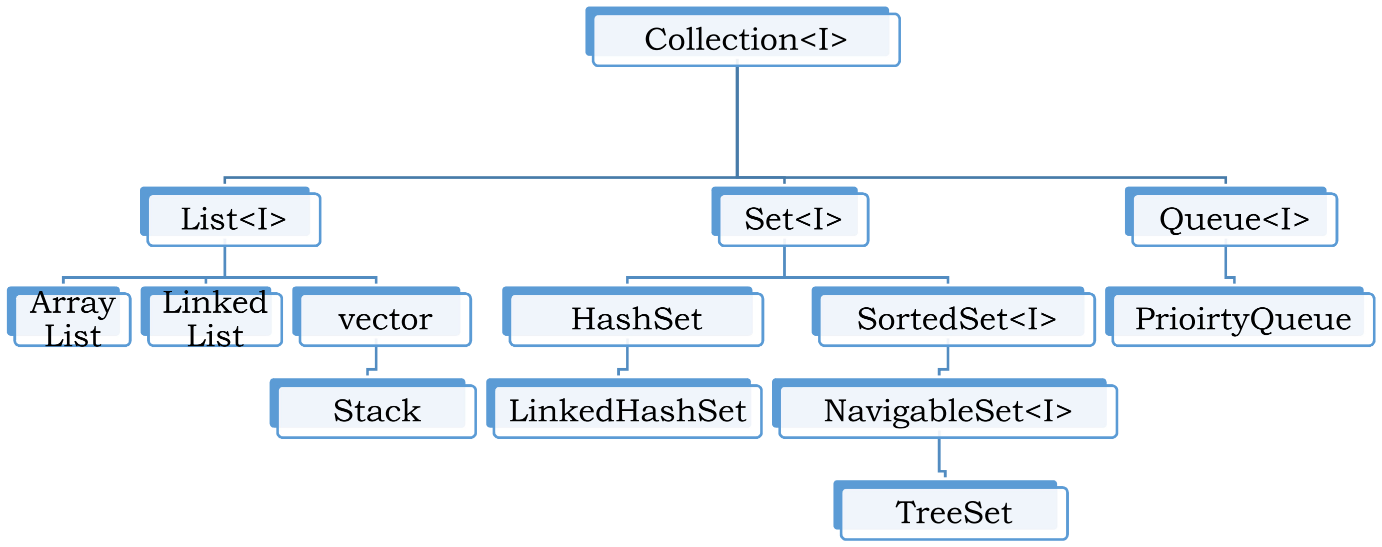
- What is an Array
  - An Array is collection of homogenous Elements.
  - Every element in an array is identified with an Index or Position
- Limitations of Array
  - Array is Fixed in Size.
  - Homogenous Elements.
  - No Underlying Data structure.
- What is Collection
  - Group of Individual objects as a single Entity.
- What is Collection Framework
  - A collections framework is a unified architecture for representing and manipulating collections.
  - All collections frameworks contain the following:
    - **Interfaces:**
      - These are abstract data types that represent collections.
      - Interfaces allow collections to be manipulated independently of the details of their representation.
      - In object-oriented languages, interfaces generally form a hierarchy.
    - **Implementations i.e. Classes:**
      - These are the concrete implementations of the collection interfaces.
      - In essence, they are reusable data structures.
    - **Algorithms:**

- These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.
- The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

How the Collection Hierarchy is designed?



What are the interfaces involved in Collection framework?

The following are the NINE interfaces of Collection Frame work

1. Collection
2. List
3. Set
4. Sortedset
5. Navigableset

6. Queue
7. Map
8. Sortedmap
9. Navigable Map

What is the difference between Collection interface and Map interface?

List of classes under Collection hierarchy is used to manage the list of values

Ex:

- List of courses → {c,c++,Java}
- List of Students
- List of Products

Map is used to manage the data in a <key,value> pair

Examples are:

- List of students where each student is identified using a key called HTNO <htno,studentInfo>

Htno	Studeninformation
101	Name: S1 Course : C
102	Name: S2 Course : CPP
103	Name: S3 Course : Java

- List of Employees where each employee is identified using a key called Employee Id <employeeid,employeeInfo>

```
package collection.app1;

import java.util.ArrayList;

import java.util.Collection;

import java.util.HashMap;

import java.util.Map;

public class CollectionVsMap {

public static void main(String[] args)
{

Collection<String> courseList = new
ArrayList<>();

/* Inserting the elements into
Collection*/

courseList.add("C");

courseList.add("CPP");

courseList.add("Java");

/* display the elements from
collection */
System.out.println(courseList);

Map<Integer, String> numToWords = new
HashMap<>();

/* Insert the pair(key,value) into Map
*/

numToWords.put(1, "One");

numToWords.put(2,"Two");

numToWords.put(3,"Three");

/* display the pairs from Map */

System.out.println(numToWords);

}

}
```

Collection Hierarchy

- Is used to manage the list of values
- This interface extends to THREE interfaces
    - List
    - Set
    - Queue

Below are some of the methods in Collection Interface

No	Method Prototype	Description
1	public int size()	Returns the number of elements in a given collection.
2	public void clear()	Clears the collection by removing all the elements from the collection.
3	public boolean add(E e)	Inserts an element e in the collection.
4	public boolean addAll(Collection< ?extends E> c)	Insert all the elements in the collection given by c into the collection.
5	public boolean remove(Object element)	Delete element given by 'element' from the collection.
6	public boolean removeAll(Collection< ?>c)	Remove the collection c from the collection.
7	default boolean removeIf(Predicate< ? super E> filter)	Remove all the elements that satisfy the given predicate 'filter' from the collection.
8	public boolean retainAll(Collection< ?> c)	Delete all elements from the collection except those in the specified collection c.
9	public Iterator iterator()	Return iterator for the collection.
10	public Object[] toArray()	Convert the collection into an array.
11	public < T> T[] toArray(T[] a)	Convert the collection into an array with a specified return type.
12	public boolean isEmpty()	Return if collection is empty or not.
13	public boolean contains(Object element)	Check if a collection contains the given element (Search operation).
14	public boolean containsAll(Collection< ?>c)	Check if collection contains specified collection c inside it.
15	default Spliterator spliterator()	Returns spliterator over a specified collection.
16	public boolean equals(Object element)	Used for matching two collections.

17	default Stream parallelStream()	Returns parallel stream using the collection as source.
18	default Streamstream()	Returns sequential stream using the collection as source.
19	public int hashCode()	Returns numeric hashcode of the collection.

```
package collection.app1;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Iterator;

public class MethodsInCollection {

public static void main(String[] args)
{

Collection<String> courseList = new
ArrayList<>();

/* Inserting the elements into
Collection*/

courseList.add("C");

courseList.add("CPP");

courseList.add("Java");
```

```
/* display the elements from
collection */

System.out.println(courseList);

System.out.println("Number of
elements: " +

courseList.size());

/* Creating duplicate list */

Collection<String> duplicateList =

new ArrayList<>();

duplicateList.addAll(courseList);

courseList.clear();

System.out.println("Number of elements
in course List: " +

courseList.size());

System.out.println("Number of elements
in duplicate List: " +

duplicateList.size());
duplicateList.remove("C");

System.out.println(duplicateList);

/* retreive the individual element
from list */

Iterator<String> courseIterator =
duplicateList.iterator();

while(courseIterator.hasNext()) {

System.out.println(courseIterator.next
());

}

System.out.println("Is C Available?" +

duplicateList.contains("C"));

System.out.println("Is Java
Available?" +

duplicateList.contains("Java"));

}

}
```

What is the difference between List, Set and Queue?

List	Set	Queue
Manages the list of elements	Manages the list of elements	Manages the list of elements
Insertion order is preserved	NO Insertion order	Every element has his own priority number
Duplicates are allowed	Won't allow Duplicate values	Allows Duplicate Values
We can perform insert operation anywhere (beginning, ending, middle) in List and Set		Insertion -→ rear end Removal -→ front
Allows Null Values		Won't allow Null values. If we try to save Null value, then it throws NULL POINTER EXCEPTION

```
package collection.app1;
import java.util.ArrayList;
import java.util.Deque;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Queue;

import java.util.Set;

public class ListVsSetVsQueue {

public static void main(String[] args)
{

List<Integer> numList = new
ArrayList<>();

numList.add(10);

numList.add(5);

numList.add(1);

numList.add(15);

numList.add(8);

numList.add(null);

numList.add(null);

numList.add(10);

numList.add(5);
```

```
System.out.println("List Contains...."
+ numList);

Set<Integer> numSet = new HashSet<>();

numSet.add(10);

numSet.add(5);

numSet.add(1);

numSet.add(15);

numSet.add(8);

numSet.add(null);

numSet.add(null);

numSet.add(10);

numSet.add(5);

System.out.println("Set Contains...."
+ numSet);

Queue<Integer> numQueue = new
PriorityQueue<>();

numQueue.add(10);
numQueue.add(5);

numQueue.add(1);

numQueue.add(15);

numQueue.add(8);

//numQueue.add(null);

//numQueue.add(null);

numQueue.add(10);

numQueue.add(5);

System.out.println("Queue
Contains...." + numQueue);

}

}
```

List

- The list interface inherits the collection interface.
- List interface contains the data structures that are used to store ordered data or collection of objects.
- These data structures are of a list type.
- These data structures implementing the list interface may or may not have duplicate values.

- List interface contains the methods that are used to access, insert or remove elements from the list-objects using Index.
- Various classes that implement the List interface are as follows:
    - ArrayList
    - LinkedList
    - Vector
    - Stack

Array List - Class

- Underlying Data structure is Resizable or Growable array.
- Insertion order is preserved
- NULL values are allowed.
- Hetrogeneous Objects are allowed.
- Duplicates are allowed.
- Constructor
- **ArrayList**()
    - Constructs an empty list with an initial capacity of ten.
    - Once it was reached to the maximum capacity then it uses following formula:
    - **ArrayList**(int initialCapacity)
    - Constructs an empty list with the specified initial capacity.
    - **ArrayList**(Collection c)
    - Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Vector - Class

- Underlying Data structure is Resizable or Growable array.
- Insertion order is preserved

- NULL values are allowed.
- Hetrogeneous Objects are allowed.
- Duplicates are allowed.
- Every method in Vector is Synchronized. So, it is ThreadSafe.
- Constructors in Vectors
  - **Vector**() Constructs an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.
  - **Vector**(Collection c) Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.
  - **Vector**(int initialCapacity) Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
  - **Vector**(int initialCapacity, int capacityIncrement) Constructs an empty vector with the specified initial capacity and capacity increment.

Similarity between ArrayList and Vector

Both maintains Array as a Data structure

Difference between vectors and arraylist

Array List	Vector
ArrayList increases by half of its size when its size is increased.	Vector doubles the size of its array when its size is increased.
Not Thread Safe No methods in the array list are synchronized	Thread Safe Every method defined in the vector is synchronized
Non Legacy in JDK1.2 Version	Legacy Collection from JDK1.0

ArrayList is way faster than Vector.	Since Vector is synchronized and thread-safe it pays price of synchronization which makes it little slow.

<u>Stack</u>

- It is a sub-class of Vector
- Follows LIFO mechanism
- Defined as:
  - public class Stack<E> extends Vector<E>
- **Constructor is:**
  - **Stack**() Creates an empty Stack.
- **Stack Methods**
  - boolean **empty**()
    - Tests if this stack is empty.
  - Object **peek**()
    - Looks at the object at the top of this stack without removing it from the stack.
  - Object **pop**()
    - Removes the object at the top of this stack and returns that object as the value of this function.
  - Object **push**(E item)
    - Pushes an item onto the top of this stack.
  - int **search**(Object o)

- Returns the 1-based position where an object is on this stack.

Linked List - Class

- Underlying Data structure is DOUBLE LINKED LIST.
- Insertion order is preserved
- NULL values are allowed.
- Heterogeneous Objects are allowed.
- Duplicates are allowed.
- Preferable when the Insert and delete operations are in the middle.
- Constructor and Description
  - **LinkedList**() Constructs an empty list.
  - **LinkedList**(Collection c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
- Methods in Linked List
  - Void addFirst(Object o)
  - Void addLast(Object o)
  - Object removeFirst();
  - Object removeLast();
  - Object getFirst();
  - Object getLast();

Make synchronized methods

- Under list Interface,
  - the classes LinkedList and ArrayList are not synchronized, to make them Synchronized use
  - public static List synchronizedList(List);

- For example:
    - ArrayList al = new ArrayList();
    - List l1 = Collections.synchronizedList(al);
    - LinkedList ll = new LinkedList();
    - List l2 = Collections.synchronizedList(l2);

- ArrayList and Vector implements Random Access interface but LinkedList won't implements Random access interface
- All three classes implements:
    - Serializable
    - Clonable
-

Note:

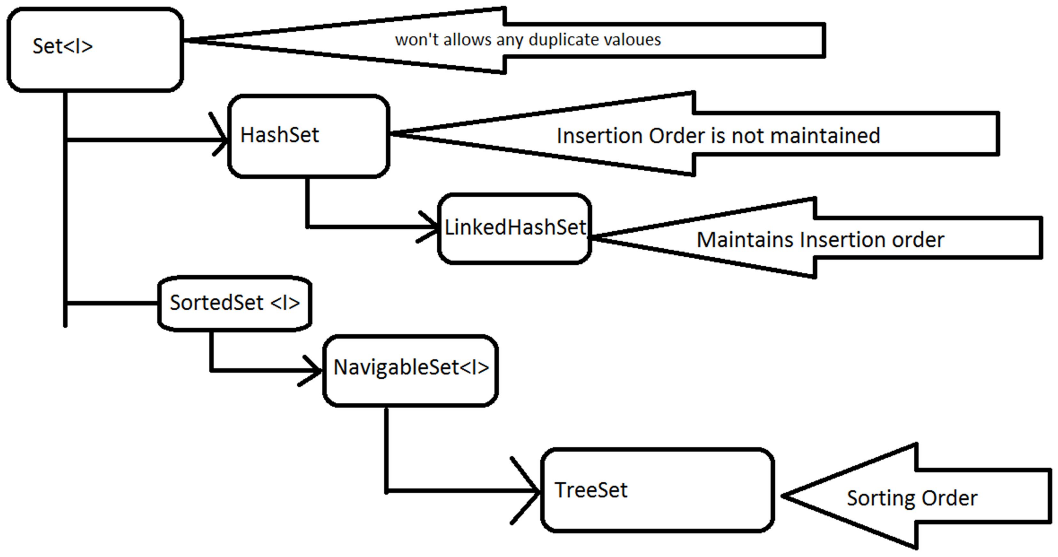
- Collection framework can be used to Store and transfer the objects.
- To do this requirement, every collection framework implements two interfaces:
    - Serializable
    - Clonable

ArrayList	Linked List	Vector
Data Structure : Array	Data Structure : Double Linked List	Data Structure : Array
Focus is on Retrieving the Data randomly	When ever we want to perform insert or delete operations, then Linked List is preferred.	Focus is on Retrieving the Data randomly
All methods in Arraylist are not synchronized	All methods in Linked list are not synchronized	All methods in Vector are synchronized

Not a Thread Safe	Not a Thread Safe	Thread Safe
Default size is 10 Incremental Factor: 3/2*currentsize + 1	No Size – NA --	Default size is 10 Incremental Factor: Double
Version: 1.2	Version 1.2	Version 1.0 → Legacy Member

Set

- The set interface is a part of the java.util package and extends from the collection interface.
- Set is a structure that doesn't allow the collection to have duplicate values and also more than one null value.
- The following classes implement the set interface.
    - HashSet
    - LinkedHashSet
    - TreeSet


Set<I>
won't allows any duplicate valoues
HashSet
Insertion Order is not maintained
LinkedHashSet
Maintains Insertion order
SortedSet <I>
NavigableSet<I>
TreeSet
Sorting Order

HASH SET

- CLASS IMPLEMENTED IN 1.2
- FOLLOWS HASH DATA STRUCTURE
- **constructors in Hashset**
  - **HashSet**()
    - Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).
  - **HashSet**(Collection<? extends E> c)
    - Constructs a new set containing the elements in the specified collection.
  - **HashSet**(int initialCapacity)

- Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).

– **HashSet**(int initialCapacity, float loadFactor)

- Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.

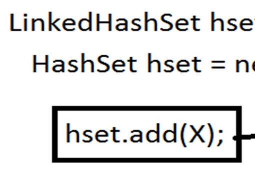
LinkedHashSet

- Child class of HashSet
- Introduced in 1.4
- Insertion order is preserved
- Hashtable and doublelinkedlist are the data structures
- LINKED HASHSET IS THE BEST APPLICATION FOR CACHE RELATED APPLICATIONS WHERE DUPLICATES ARE NOT ALLOWED.

DIFFERENCES BETWEEN HASHSET AND LINKED HASHSET

HASHSET	LINKEDHASHSET
UNDERLYING DATA STRUCTURE IS HASH TABLE	UNDERLYING DATA STRUCTURE IS HASH TABLE AND LINKED LIST
INSERTION ORDER IS NOT PRESERVED	INSERTION ORDER IS PRESERVED
INTRODUCED IN JAVA 1.2	INTRODUCED IN JAVA 1.4

How Hashset or LinkedHashset identifies the duplicate elements?



Note: That is the reason, why we have to override hashcode() method whenever we are overriding equals method.

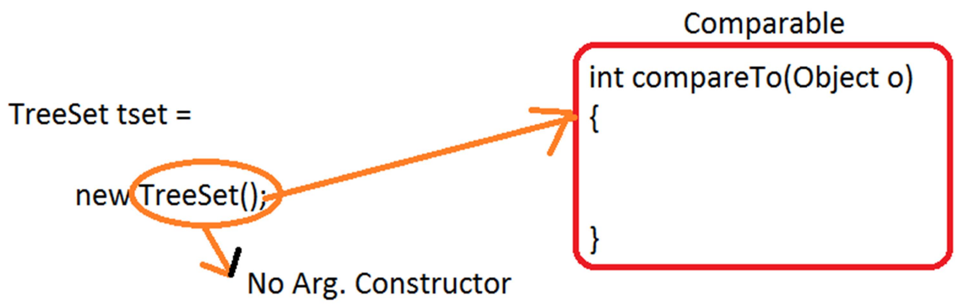
SORTED SET

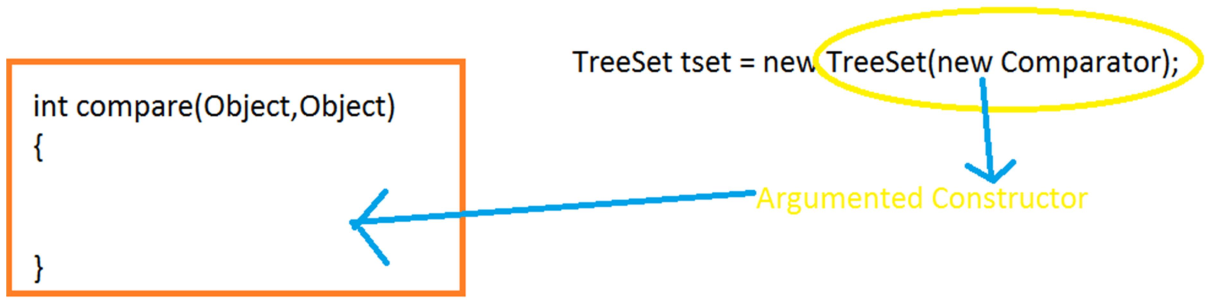
- IT IS AN INTERFACE
- IMPLEMENTED IN JAVA 1.2
- FOLLOWS DEFAULT ORDERING OR CUSTOMIZING ORDERING.

TREE SET

- UNDERLYING DATA STRUCTURE IS BALANCED TREE
- DUPLICATES ARE NOT ALLOWED.
    - IF TRYING TO STORE THEN ADD METHOD RETURNS FALSE.
- HETROGENEOUS ELEMENTS ARE NOT ALLOWED WHEN NATURAL ORDERING IS FOLLOWING IF THE ORDER IS CUSTOMIZED THEN WE CAN PROVIDE HETROGENOUS.
    - IF TRIES TO STORE RAISES CLASS CAST EXCEPTION
- INSERTION ORDER IS NOT PRESERVED.
- Based on sorting order

- Null is allowed only once.
  - IF NULL IS TRYING TO STORE IN NON-EMPTY SET THEN NULL POINTER EXCEPTION IS ARAISED
- Constructor and Description
  - **TreeSet**() Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
  - **TreeSet**(Collection<? extends E> c) Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.
  - **TreeSet**(Comparator<? super E> comparator) Constructs a new, empty tree set, sorted according to the specified comparator.
  - **TreeSet**(SortedSet<E> s) Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.




int compare(Object,Object)
{
}
TreeSet tset = new TreeSet(new Comparator);
Argumented Constructor

How TreeSet arranges the data in Natural Ordering or customized ordering?

Treeset takes support of (1) Comparable or (2) Comparator to arrange the data either in Natural Ordering or User defined ordering.

<u>Differences between Comparable and Comparator</u>

Comparable

- Defined in **java.lang** package
- One method called **compareTo()**
- Used to sort the keys using natural or default ordering.
- All Wrapper class and string class are implemented by Comparable Interface.
- Drawbacks of Comparable
- Can be applied to classes that are implemnting **Comparable** Interface.
- Follows natural or default ordering.

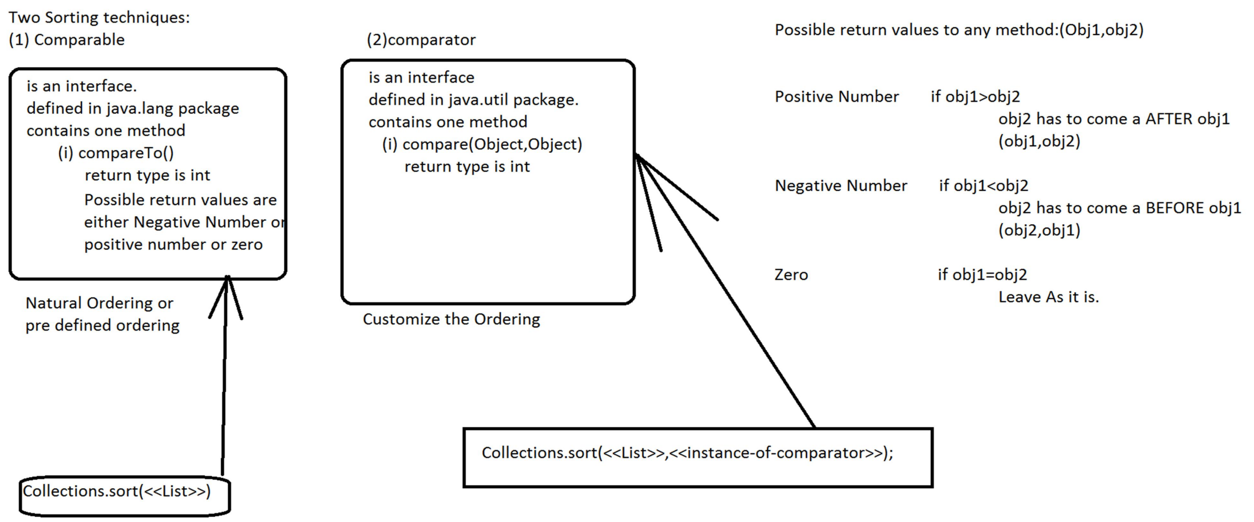
Comparator Interface

- Defined in java.util package.
- Used for customized sorting order.
- Contains two Methods:
  - public int compare(Object o1, Object o2)
  - public boolean equals()
- In order to define compare Method, the class should implement Comparator Interface.
- Implementation of equals() method in this interface is optional. Because it is already in object class.

examples

- Class MyComparator implements Comparator
- {
-     public int compare(Object o1,Object o2)

- {
- Integer i1 = (Integer)o1;
- Integer i2 = (Integer)o2;
- .
- .
- .
- }
- return i1.compareTo(i2); // ascending order
- return - i1.compareTo(i2); // descending order
- return i2.compareTo(i1); // descending order
- return -i2.compareTo(i1); // Ascending order
- return -1 // reverse of the Insertion order
- return +1 // insertion order is preserved
- return 0 // first key is stored.



Two Sorting techniques:
(1) Comparable

is an interface.
defined in java.lang package
contains one method
(i) compareTo()
return type is int
Possible return values are either Negative Number or positive number or zero

Natural Ordering or pre defined ordering

Collections.sort(<<List>>)

(2)comparator

is an interface
defined in java.util package.
contains one method
(i) compare(Object,Object)
return type is int

Customize the Ordering

Collections.sort(<<List>>,<<instance-of-comparator>>);

Possible return values to any method:(Obj1,obj2)

Positive Number	if obj1>obj2	obj2 has to come a AFTER obj1 (obj1,obj2)
Negative Number	if obj1<obj2	obj2 has to come a BEFORE obj1 (obj2,obj1)
Zero	if obj1=obj2	Leave As it is.

CompareTo() is overriden in all Predefined data types and String Class.
To deal with Userdefined Data types like Student,Empoyee...., we have to override compare() method in the respective class by implementing Comparable interface

How comparable arranges the below data in Natural ordering?

5,1,15,20,1

Element	CompareTo()	Return Value	Ordered Elements
5			5
1	1.compareTo(5) → 1<5	-1 (Before 5)	1,5
15	15.compareTo(1) → 15>1	+1 (After 1)	
	15.compareTo(5) → 15 > 5	+1 (After 5)	1,5,15
20	20.compareTo(5) → 20 > 5	+1 (After 5)	
	20.compareTo(15) →20 > 15	+1 (After 15)	1,5,15,20
1	1.compareTo(5) → 1 < 5	-1 (Before 5)	
	1.compareTo(1) → 1==1	0 (Leave As it is)	1,1,5,15,20

NAVIGABLE SET

- It is an Interface
- Implemented in Java 1.6
- Child interface of Sorted set.
- Set s = Collections.synchronizedSet(new HashSet(...));
- Set s = Collections.synchronizedSet(new LinkedHashSet(...));
- SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

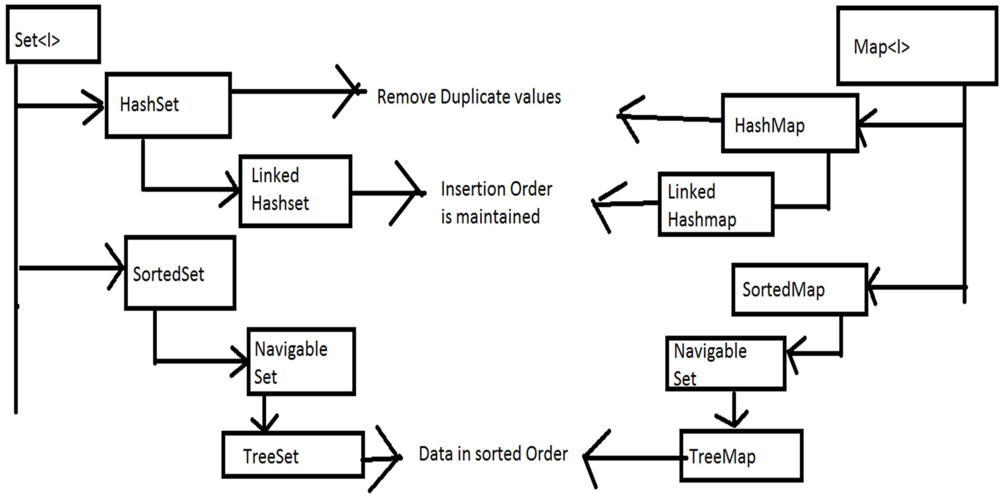
Queue Interface

- public interface Queue<E> extends Collection<E>
- Follows FIFO mechanism
- Null values are not allowed. If given, raises NullPointerException
- Methods of Queue Interface
- boolean **add**(E e)
  - Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
- boolean **offer**(E e)
  - Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
- E **element**()
- Retrieves, but does not remove, the head of this queue. Throws Exception when element is not available
- E **peek**()
  - Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- E **poll**()
  - Retrieves and removes the head of this queue, or returns null if this queue is empty.
- E **remove**()
  - Retrieves and removes the head of this queue. Throws Exception

Priority Queue

- An unbounded priority queue based on a priority heap

- It follows Natural Ordering defined at comparator interface or definition that was provided at compile time.
- Null values are not allowed here.
- constructors
- Methods in Dequeue



Difference between Collection and Map

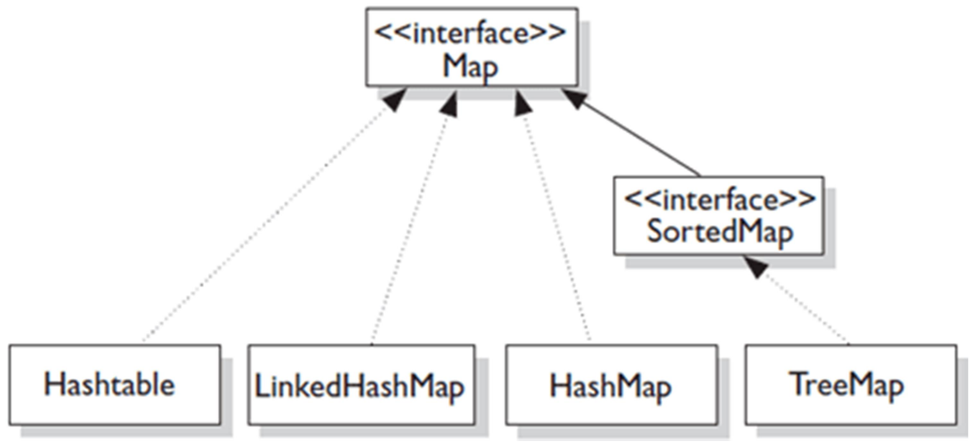
Collection	Map
Maintains the list of values {10,20,30,…….}	Maintains the data in the form of <key,value> pair. Where key is a unique data used to

{“stud1”,”stud2”,…….}	identify value. {<101,”stud1”>,<102,”stud2”,………}
Some of the methods are: 1. add() --- insert an element 2. contains() ---- check whether an element is existed or not 3. remove() --- to delete an element	Some of the methods are: 1. put(k,v) --- insert an pair into map 2. containsKey(k) 3. containsValue(v) 4. remove(k)

MAP

- If we want to represent group of objects in a key – value pair, then we must go to MAP interface.
- Duplicate keys are not allowed.
- But values may duplicate.
- Combination of key and its associated value is called as an ENTRY.
- **Commonly used methods of Map interface:**
- **public Object put(object key,Object value):** is used to insert an entry in this map.
- **public void putAll(Map map)**:is used to insert the specifed map in this map.
- **public Object remove(object key):i**s used to delete an entry for the specified key.
- **public Object get(Object key):**is used to return the value for the specified key.
- **public boolean containsKey(Object key)**:is used to search the specified key from this map.

- **public boolean containsValue(Object value)**:is used to search the specified value from this map.
- **public Set keySet()**:returns the Set view containing all the keys.
- **public Set entrySet()**:returns the Set view containing all the keys and values



<u>HASHMAP</u>

- DATASTRUCTURE IS HASH TABLE
- DUPLICATE KEYS ARE NOTALLOWED
- INSERTION ORDER IS NOT PRESERVED AND VALUES ARE STORED BASED ON HASH CODE OF THE KEY
- NULL KEY IS ALLOWED ONLY ONCE AND NULL VALUES ARE NOT ALLOWED.
- HETROENEOUS KEYS AD VALUES ARE ALLOWED
- HASH MAP METHODS ARE NOT SYNCHRONIZED
- NULL IS ALLOWED FOR KEY AS WELL AS VALUE
- Introduced in java 1.2
- HASH TABLE METHODS ARE SYNCHRONIZED
- NULL NOT ALLOWED IN KEY AS WELL AS VALUE

- Legacy Member of Java

ie. Introduced in Java 1.1

Map m = Collections.synchronizedMap(new HashMap(...));

Linked HashMap

- Child interface of HASHMAP
- HASH MAP WON'T PRESERVES THE ORDER OF INSERTION
- INTRODUCED IN 1.2
- LINKED HASHMAP PRESERVES THE ORDER OF INSERTION
- INTRODUCED IN 1.4

SortedMap

NAVIGABLE MAP

- It is an Interface
  - ALL THE METHODS ARE USED FOR NAVIGATION PURPOSE
- Implemented in Java 1.6
- Child interface of Sorted Map.

Method	Purpose
ceilingKey(e)	Lowest key which is >=e
higherKey(e)	Lowest key which is >e
floorKey(e)	Highest key which is <=e
lowerKey(e)	Highest key which is <e

pollFirstEntry()	Removes and returns the first key
pollLastEntry()	Removes and returns last key
descedingMap()	Returns the NavigabeMap in Descending order

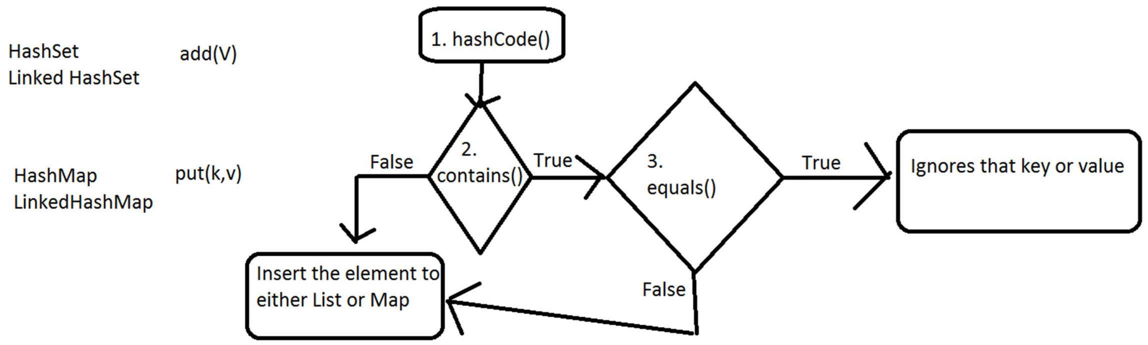
TreeMap

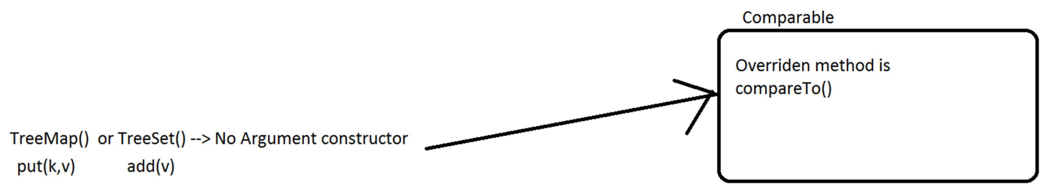
- It stores key-value pairs similar to like HashMap.
- It allows only distinct keys. Duplicate keys are not possible.
- It cannot have null key but can have multiple null values.
- It stores the keys in sorted order (natural order) or by a Comparator provided at map creation time.
- It provides guaranteed log(n) time cost for the containsKey, get, put and remove operations.
- It is not synchronized. Use Collections.synchronizedSortedMap(new TreeMap()) to work in concurrent environment.
- The iterators returned by the iterator method are fail-fast.

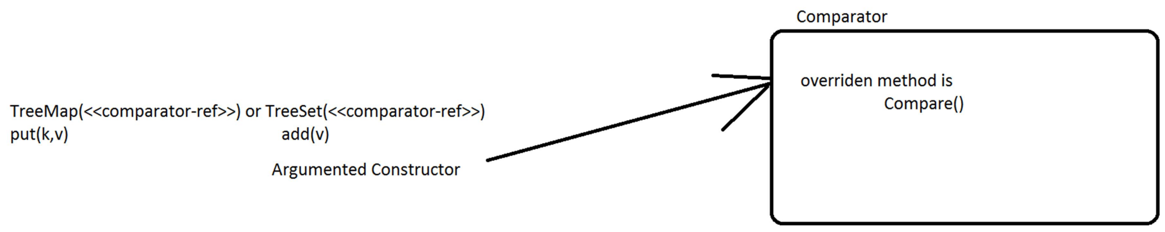
What is the difference beween different classes of Map?

HashMap	TreeMap	LinkedHashmap	HashTable
Null keys and values are allowed	Only null values allowed.	Null keys and values allowed.	It won't allow null keys and values.
Not synchronized	Not synchronized	Not synchronized	synchronized
There is no guarantee to maintain order in the iteration	Sorting will be done based on natural order.	Insertion order will be maintained	Insertion order not maintained.

How Set or Map identifies duplicate elements?


HashSet
Linked HashSet
add(V)
HashMap
LinkedHashMap
put(k,v)
1. hashCode()
2. contains()
False
True
3. equals()
True
Ignores that key or value
False
Insert the element to either List or Map


Comparable
Overriden method is compareTo()
TreeMap() or TreeSet() --> No Argument constructor
put(k,v) add(v)


Comparator
overriden method is Compare()
TreeMap(<<comparator-ref>>) or TreeSet(<<comparator-ref>>)
put(k,v) add(v)
Argumented Constructor

CURSORS IN JAVA

1. ENUMERATION --- 1.1
2. ITERATOR -- 1.2
3. LIST ITERATOR -- 1.2

ENUMERATION

- APPLICABLE ONLY FOR LEGACY CLASSES.
- EXAMPLE:
  - VECTOR
  - STACK
- LIMITATIONS OF ENUMERATION
  - Can be applied for Legacy Methods. i.e. not a Universal Cursor.
  - Read only operations are only possible.

ITERATOR

- IT IS A UNIVERSAL CURSOR.
- ALLOWS DELETE OPERATION WHILE ITERATING.
- LIMITATIONS OF ITERATOR

  - UNI DIRECTIONAL I.E. FORWARD DIRECTION
  - ONLY READ AND DELETIONS ARE POSSIBLE BUT ADDITION AND MODIFICATION IS NOT POSSIBLE.

LIST ITERATOR

- CHILD INTERFACE OF ITERATOR.
- IT IS BIDIRECTIONAL.
- ALLOWS ADDITION ,MODIFICATION AND DELETE OPERATIONS WHILE ITERATING.

LEGACY MEMBERS IN JAVA

1. ENUMERATON -- INTERFACE
2. DICTIONARY -- ABSTRACT CLASS
3. VECTOR -- CLASS
4. STACK -- CLASS
5. HASHTABLE -- CLASS
6. PROPERTIES -- CLASS