Lesson:

Collection and Map API in Java

PW SKILLS

List of Concepts Involved:

- Why Collection?
- Collection Hierarchy
- ArrayList
- LinkedList
- ArrayDeque
- PriorityQueue
- TreeSet
- HashSet
- LinkedHashSet
- Iterator , List Iterator
- Legacy classes and Enumeration
- Introduction to Map in Java
- Map Hierarchy
- HashMap
- Other In-Built classes and Inbuilt methods under Map Hierarchy

Why Collection?

1. They are growable in nature(we can increase and decrease)
2. They can hold both heterogeneous and homogeneous data elements
3. Every collection class is implemented using some standard data structure, so ready methods are available, as a programmer we need to implement rather we should just know how to call those methods.

Which one is prefered over Arrays and Collections?

Arrays is prefered, because performance is good.
Collections is not prefered because

1. List l=new ArrayList(); // default: 10 locations
   if 11th element has to added, then
   a. create a list with 11 locations
   b. copy all the elements from the previous collection
   c. copy the new reference into reference variable
   d. call the garbage collector and clean the old memory.

Note: To get something we need to compromise something, so if we use Collections performance is not upto the mark.
Array is language level concept(memory wise it is not good, performance is high)
Collection is API level(memory wise it is good,performance is low)

Difference b/w Arrays and Collection

Arrays => It is used only when Array size is fixed
Collection => It is used only when size is not fixed(dynamic)

Arrays => memory wise not recommended to use.
Collection => memory wise recommended to use.

Arrays => Performance wise recommended to use.
Collection => Performance wise it is recommended to use.

Arrays => It can hold only homogeneous objects
Collection => It can hold both heterogenous and homogenous Objects

Arrays => We can hold both primitive values and Objects
eg: int[] arr=new int[5];
Integer[] arr=new Integer[5];
Collection => It is capable of holding only objects not primitive types.

Arrays => It is not implemented using any standard data structure, so no ready made methods For our requirement,it increases the complexity of programming.

Collection => It is implemented using standard data structure, so ready made methods are available for our requirement,it is not complex.

What is a Collection?

In Order to represent a group of individual object as a single entity then we need to go for Collection.

CollectionFramework

Group of classes and interface, which can be used to represent a group of individual object as a single entity, then we need to go for "CollectionFramework".

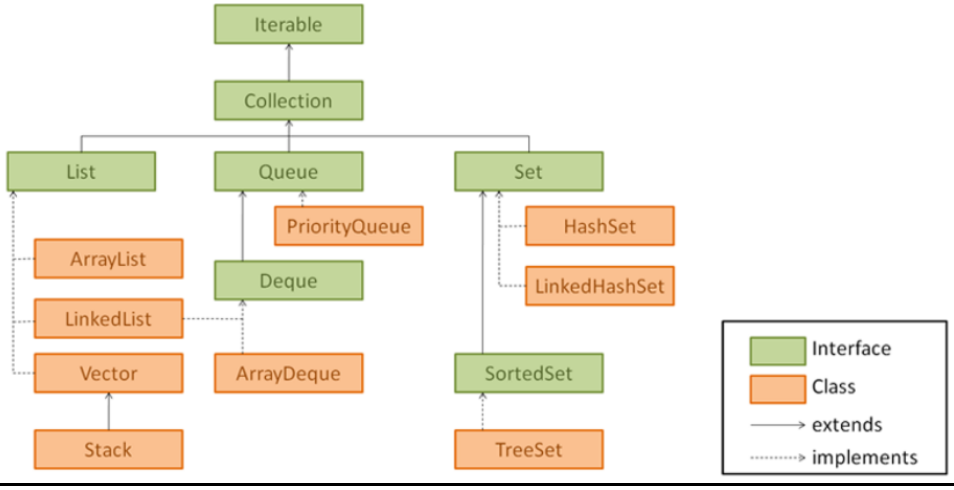
Java C++
Collection => container
CollectionFramework => STL(standard template library)

Collection

1. In order to represent a group of individual object, then we need to go for "Collection".
2. It is a root interface of collection framework
3. All the commonly used method required for all the collections are a part of Collection(I).

Note: There is no concrete class which would implement the interface Collection(I) directly.

Collection Hierarchy



To know more information about the framework,then we need to know the specification(interface)
9 key interfaces of Collection framework

1. Collection(I)
2. List(I)
3. Set(I)
4. SortedSet(I)
5. NavigableSet(I)
6. Queue(I)

MAP(I) → We will see in future lecture

7. Map(I)
8. SortedMap(I)
9. NavigableMap(I)

ArrayList(C)

1. DataStructure: GrowableArray /Sizeable Array
2. Duplicates are allowed through index
3. insertion order is preserved through index
4. Heterogenous objects are allowed.
5. null insertion is also possible.

Constructors

a. ArrayList al=new ArrayList()
Creates an empty ArrayList with the capacity to 10.

a. if the capacity is filled with 10, then what is the new capacity?
new capacity= (currentcapacity * 3/2)+1
so new capacity is =16,25,38,.....
b. if we create an ArrayList in the above mentioned order then it would result in performance issue.
c. To resolve this problem create an ArrayList using the 2nd way approach.

b. ArrayList al=new ArrayList(int initialCapacity)
c. ArrayList l=new ArrayList(Collection c)
It is used to create an equivalent ArrayList Object based on the Collection Object

When to use ArrayList and when not to use?

ArrayList => it is best suited if our frequent operation is "retrieval operation",because it implements RandomAccess interface.

ArrayList => it is the worst choice if our frequent operation is "insert/deletion" in the middle because it should perform so many shift operations.To resolve this problem we should use "LinkedList".

LinkedList

- Memory management is done effectively if we work with LinkedList.
- memory is not given in continuous fashion.

a. DataStructure is :: doubly linked list
b. heterogenous objects are allowed

c. null insertion is possible
d. duplicates are allowed

Usage

1. If our frequent operation is insertion/deletion in the middle then we need to opt for "LinkedList".

```
LinkedList l=new LinkedList();
l.add(a);
l.add(10);
l.add(z);
l.add(2,'a');
l.remove(3);
```

2. LinkedList is the worst choice if our frequent operation is retrieval operation

Constructors

a. LinkedList l=new LinkedList();
It creates an empty LinkedList object.
b. LinkedList l=new LinkedList(Collection c);
To convert any Collection object to LinkedList.

ArrayDeque

- The ArrayDeque class implements the Deque interface.
- It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.
- ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

PriorityQueue

- The PriorityQueue class implements the Queue interface.
- It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

TreeSet

- Underlying Data Structure: BalancedTree
- duplicates : not allowed
- insertion order : not preserved
- heterogeneous element: not possible,if we try to do it would result in "ClassCastException".
- null insertion : possible only once
- Implements Serializable and Cloneable interface,but not RandomAccess.
- All Objects will be inserted based on "some sorting order" or "customised sorting order".

Constructor

TreeSet t=new TreeSet();//All objects will be inserted based on some default natural sorting order.

TreeSet t=new TreeSet(Comparator); //All objects will be inserted based on some customized sorting order.

Set

- 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 and Hence we have to Use Only Collection Interface Methods

HashSet

1. Duplicates are not allowed,if we try to add it would not throw any error rather it would return false.
2. Internal DataStructure: Hashtable
3. null insertion is possible.
4. heterogeneous data elements can be added.
5. If our frequent operation is search, then the best choice is HashSet.
6. It implements Serializable,Cloneable, but not random access.

Constructors

HashSet s=new HashSet(); Default initial capacity is 16
Default FillRation/load factor is 0.75

Note: In case of ArrayList, default capacity is 10, after filling the complete capacity then a new ArrayList would be created.
In the case of HashSet, after filling 75% of the ratio only new HashSet will be created.

HashSet s=new HashSet(int initialCapacity);//specified capacity with default fill ration=0.75
HashSet s=new HashSet(int initaliCapacity,float fillRatio)
HashSet s=new HashSet(Collection c);

LinkedHashSet

- It is the child class of "HashSet".
- DataStructure: HashTable + linkedlist
- duplicates : not allowed
- insertion order: preserved
- null allowed : yes

All the constructors and methods which are a part of HashSet will be a part of "LinkedHashSet",but except "insertion order will be preserved".

Difference b/w HashSet and LinkedHashSet

HashSet => underlying data structure is "HasTable"
LinkedHashSet => underlying data structure is a combination of "Hashtable + "linkedlist" .
HashSet => Duplicates are not allowed and insertion order is not preserved
LinkedHashSet => Duplicates are not allowed,but insertion order is preserved.
HashSet => 1.2V
LinkedHashSet => 1.4v

The 3 Cursors of Java

- If we want to get Objects One by One from the Collection then we should go for Cursors.
- There are 3 Types of Cursors Available in Java.

1. Enumeration
2. Iterator

3. ListIterator

Enumeration:

We can Use Enumeration to get Objects One by One from the Collection.
We can Create Enumeration Object by using elements().
public Enumeration elements();
Eg:Enumeration e = v.elements(); //v is Vector Object.

Methods:

1. public boolean hasMoreElements();
2. public Object nextElement();

```
import java.util.*;
public class EnumerationDemo {
        public static void main(String[] args) {
                Vector v = new Vector();
                for(int i=0; i<=10; i++) {
                        v.addElement(i);
                }
                System.out.println(v);//[0,1,2,3,4,5,6,7,8,9,10]
                Enumeration e = v.elements();
                while(e.hasMoreElements()) {
                        Integer I = (Integer)e.nextElement();
                        if(I%2 == 0)
                                System.out.println(I);//0 2 4 6 8 10
                }
                System.out.println(v);//[0,1,2,3,4,5,6,7,8,9,10]
        }
}
```

Limitations of Enumeration:

- Enumeration Concept is Applicable Only for Legacy Classes and it is Not a Universal Cursor.
- By using Enumeration we can Perform Read Operation and we can't Perform Remove Operation.

To Overcome Above Limitations we should go for Iterator.

3. ListIterator:

- ListIterator is the Child Interface of Iterator.
- By using ListIterator we can Move Either to the Forward Direction OR to the Backward Direction. That is it is a Bi-Directional Cursor.
- By using ListIterator we can Able to Perform Addition of New Objects andReplacing existing Objects. In Addition to Read and Remove Operations.
- We can Create ListIterator Object by using listIterator().
  public ListIteratorlistIterator();
  Eg:ListIteratorItr = l.listIterator(); //l is Any List Object

Methods:

- ListIteratoris the Child Interface of Iterator and Hence All Iterator Methods by Default Available to the ListIterator.

Iterator(I)
|
|
ListIterator(I)

ListIteratorDefines the following 9 Methods.

public boolean hasNext()
public Object next()
public int nextIndex()

public boolean hasPrevious()
public Object previous()
public int previousIndex()

public void remove()
public void set(Object new)
public void add(Object new)

Legacy classes

- Legacy classes refers to the older classes that were included in the early versions of Java and have since been replaced by newer, more efficient classes. One such class is Enumeration, which is a legacy interface that was used to traverse collections before the introduction of the Iterator interface.

The Legacy classes are Dictionary, Hashtable, Properties, Stack, and Vector. The Legacy interface is the Enumeration interface.

Introduction to Map in Java

Map

To represent a group of individual objects as a key value pair then we need to opt for Map(I).
It is not a child interface of Collection.

- If we want to represent a group of Objects as a key-value pair then we need to go for Map.
- Both keys and values are Objects only
- Duplicate keys are not allowed but values are allowed.
- Key-value pair is called "Entry".

Map interface

It contains 12 methods which is common for all the implementation Map Objects
a. Object put(Object key,Object value)
b. void putAll(Map m)
c. Object get(Object key)
d. Object remove(Object key)
e. boolean containsKey(Object key)
f. boolean containsValue(Object value)
g. boolean isEmpty()
h. int size()
i. void clear()

views of a Map

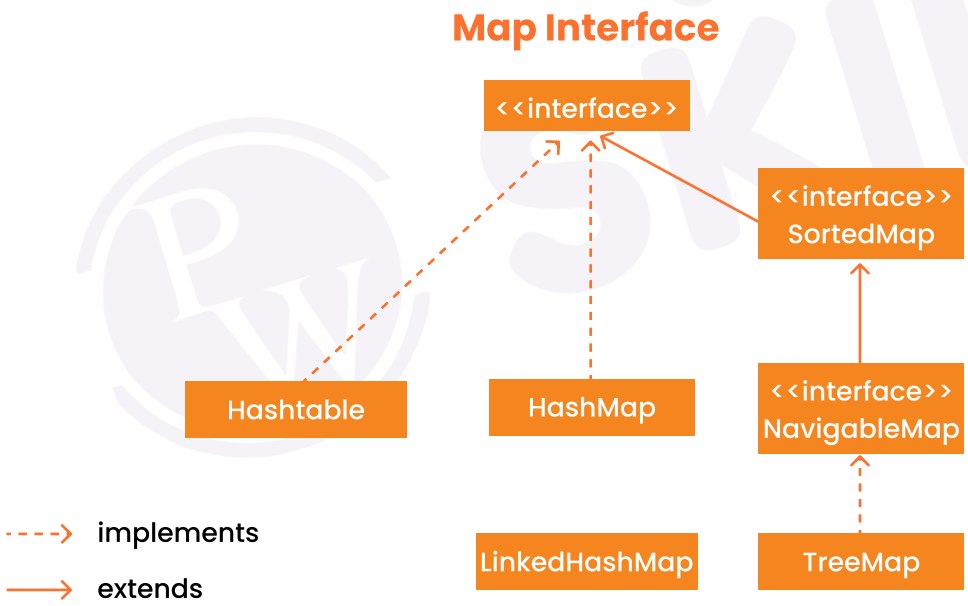
j.Set keySet()
k.Collection values()
l.Set entrySet()

Entry(I)

1. Each key-value pair is called Entry.
2. Without the existence of a Map,there can't be the existence of an Entry Object.
3. Interface entry is defined inside the Map interface.

```
interface Map{
        interface Entry{
                Object getKey();
                Object getValue();
                Object setValue(Object newValue);
        }
}
```

Map Hierarchy



HashMap

- Underlying DataStructure: Hashtable
- insertion order : not preserved
- duplicate keys : not allowed
- duplicate values : allowed
- Heterogenous objects : allowed
- null insertion : for keys allowed only once,but for values can be any no.
- implementation interface: Serializable,Cloneable.

Difference b/w HashMap and Hashtable

HashMap => All the methods are not synchronized.
Hashtable => All the methods are synchronised.

HashMap => At a time multiple threads can operate on an Object, so it is ThreadSafe.
Hashtable => At a time only one Thread can operate on an Object, so it is not ThreadSafe.

HashMap => Performance is high.
Hashtable => performance is low.

HashMap => null is allowed for both keys and values.
Hashtable => null is not allowed for both keys and values,it would result in NullPointerException.

HashMap => Introduced in 1.2v
Hashtable => Introduced in 1.0v

Note: By default HashMap is nonSynchronized,to get the synchronized version of HashMap we need to use synchronizedMap() of Collection class.

Constructors

1. HashMap hm=new HashMap()
   //default capacity => 16, loadfactor => 0.75
2. Hashmap hm=new HashMap(int capacity);
3. HashMap hm=new HashMap(int capacity,flot filtration);
4. HashMap hm=new HashMap(Map m);

LinkedHashMap

=> It is the child class of HashMap.
=> It is same as HashMap,but with the following difference

HashMap => underlying data structure is hashtable.
LinkedHashMap => underlying data structure is LinkedList + hashtable.

HashMap => insertion order not preserved.
LinkedHashMap => insertion order preserved.

HashMap => introduced in 1.2v
LinkedHashMap => introduced in 1.4v

SortedMap

1. It is the child interface of Map
2. If we want an Entry object to be sorted and stored inside the map,we need to use "SortedMap".

SortedMap defines few specific method like
a. Object firstKey()
b. Object lastKey()
c. SortedMap headMap(Object key)
d. SortedMap tailMap(Object key)

e. SortedMap subMap(Object obj1,Object obj2)
f. Comparator comparator()

NavigableMap (I):

- It is the Child Interface of SortedMap.
- It Defines Several Methods for Navigation Purposes.

TreeMap

1. Underlying data structure is "redblacktree".
2. Duplicate keys are not allowed,whereas values are allowed.
3. Insertion order is not preserved and it is based on some sorting order.
4. If we are depending on natural sorting order,then those keys should be homogenous and it should be Comparable otherwise ClassCastException.
5. If we are working on customisation through Comparator,then those keys can be heterogeneous and it can be NonComparable.
6. No restrictions on values, it can be heterogeneous or NonComparable also.
7. If we try to add null Entry into TreeMap, it would result in "NullPointerException".

Hashtable:

- The Underlying Data Structure for Hashtable is Hashtable Only.
- Duplicate Keys are Not Allowed. But Values can be Duplicated.
- Insertion Order is Not Preserved and it is Based on Hashcode of the Keys.
- Heterogeneous Objects are Allowed for Both Keys and Values.
- null Insertion is Not Possible for Both Key and Values. Otherwise we will get a Runtime Exception Saying NullPointerException.
- It implements Serializable and Cloneable,but not RandomAccess.
- Every Method Present in Hashtable is Synchronized and Hence Hashtable Object is Thread Safe,so best suited when we work with Search Operation.