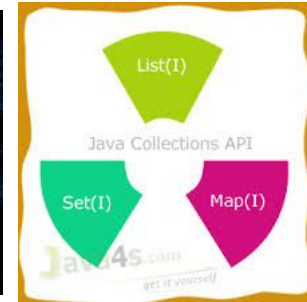


CHAPTER 11

SETS AND MAPS



SUBJECT: OOP-I
CODE: 3140705

PREPARED BY:
ASST. PROF. NENSI KANSAGARA
(CSE DEPARTMENT, ACET)



SETS:

- ❖ The set interface extends the collection interface
- ❖ The set is a collection having no duplicates elements
- ❖ The concrete classes of set are
HashSet, LinkedHashSet and TreeSet



HASH SET:

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class is not synchronized. However it can be synchronized explicitly like this:

```
Set s = Collections.synchronizedSet(new HashSet(...));
```



Points to Note about HashSet:

1. HashSet doesn't maintain any order, the elements would be returned in any random order.
2. HashSet doesn't allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
3. HashSet allows null values however if you insert more than one nulls it would still return only one null value.
4. HashSet is non-synchronized.
5. The iterator returned by this class is fail-fast which means iterator would throw `ConcurrentModificationException` if HashSet has been modified after creation of iterator, by any means except iterator's method.

SN	Modifier & Type	Method	Description
1)	boolean	<code>add(E e)</code>	It is used to add the specified element to this set if it is not already present.
2)	void	<code>clear()</code>	It is used to remove all of the elements from the set.
3)	object	<code>clone()</code>	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	<code>contains(Object o)</code>	It is used to return true if this set contains the specified element.
5)	boolean	<code>isEmpty()</code>	It is used to return true if this set contains no elements.
6)	Iterator<E>	<code>iterator()</code>	It is used to return an iterator over the elements in this set.
7)	boolean	<code>remove(Object o)</code>	It is used to remove the specified element from this set if it is present.
8)	int	<code>size()</code>	It is used to return the number of elements in the set.
9)	Splitterator<E>	<code>spliterator()</code>	It is used to create a late-binding and fail-fast Splitterator over the elements in set.

```
package javaapplicationhashset;
import java.util.*;

public class JavaApplicationHashSet {

    public static void main(String[] args) {
        char ans='y';
        int val;
        System.out.println("\n\t\t program to create HashSet");
        HashSet obj = new HashSet();
        Scanner s = new Scanner(System.in);
        System.out.println("\n Creation of HashSet");
        do
        {
            System.out.println("Enter the element");
            val = s.nextInt();
            obj.add(val);
            System.out.println("\n do you want to enter more elements?");
            String str=s.next();
            ans=str.charAt(0);
        }
        while(ans=='y');
        System.out.println("The elements are:"+obj);
    }
}
```

```
run:

      program to create HashSet

Creation of HashSet
Enter the element
10

do you want to enter more elements?
y
Enter the element
20

do you want to enter more elements?
y
Enter the element
30

do you want to enter more elements?
y
Enter the element
50

do you want to enter more elements?
n
The elements are:[50, 20, 10, 30]
BUILD SUCCESSFUL (total time: 29 seconds)
|
```



LINKEDLIST HASHSET:

LinkedHashSet is also an implementation of Set interface, it is similar to the HashSet and TreeSet except the below mentioned differences:

1. HashSet doesn't maintain any kind of order of its elements.
2. TreeSet sorts the elements in ascending order.
3. LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.


```
package javaapplicationlinkedlisthashset;
import java.util.Scanner;
import java.util.LinkedHashSet;

public class JavaApplicationLinkedListHashSet {

    public static void main(String[] args) {
        LinkedHashSet <String> s = new LinkedHashSet<>();
        s.add("aditya");
        s.add("Nensi");
        s.add("Khushi");
        s.add("Amit");
        s.add("Foram");
        s.add("Sunil");
        System.out.println(s);
        System.out.println("The size of Linked HashSet is:"+s.size());
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the name to be deleted from set:");
        String element = input.next();
        s.remove(element);
        System.out.println("The Linked HasSet is:");
        System.out.println(s);
    }
}
```

tput - JavaApplicationLinkedListHashSet (run) X

```
run:
[aditya, Nensi, Khushi, Amit, Foram, Sunil]
The size of Linked HashSet is:6
Enter the name to be deleted from set:
Amit
The Linked HasSet is:
[aditya, Nensi, Khushi, Foram, Sunil]
BUILD SUCCESSFUL (total time: 13 seconds)
```



TREE SET:

TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn't maintain any order. TreeSet allows null element but like HashSet it doesn't allow. Like most of the other collection classes this class is also not synchronized, however it can be synchronized explicitly like this:

```
SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
```

```
package javaapplicationtreeset;
import java.util.*;
import java.io.*;

public class JavaApplicationTreeSet {

    public static void main(String[] args) {
        char ans='y';
        int val;
        System.out.println("\n\t\t Program to create Tree Data structure");
        TreeSet obj= new TreeSet();
        Scanner s=new Scanner(System.in);
        System.out.println("\n Creation of Tree");
        do
        {
            System.out.println("Enter the elements");
            val = s.nextInt();
            obj.add(val);
            System.out.println("\n Do you want to enter more elements?");
            String str = s.next();
            ans=str.charAt(0);

        }while(ans=='y');
        System.out.println("The tree is:"+obj);
    }
}
```

Program to create Tree Data structure

```
Creation of Tree
Enter the elements
15

Do you want to enteer more elements?
Y
Enter the elements
2

Do you want to enteer more elements?
Y
Enter the elements
9

Do you want to enteer more elements?
Y
Enter the elements
5

Do you want to enteer more elements?
n
The tree is:[2, 5, 9, 15]
BUILD SUCCESSFUL (total time: 24 seconds)
```



COMPARING THE PERFORMANCE OF SETS AND LISTS

1) List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.

Set is an unordered collection, it doesn't maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order).

2) List allows duplicates while Set doesn't allow duplicate elements. All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.



3) List implementations: ArrayList, LinkedList etc.

Set implementations: HashSet, LinkedHashSet, TreeSet etc.

4) List allows any number of null values. Set can have only a single null value at most.

5) ListIterator can be used to traverse a List in both the directions(forward and backward) However it can not be used to traverse a Set. We can use Iterator (It works with List too) to traverse a Set.

6) List interface has one legacy class called Vector whereas Set interface does not have any legacy class.

```
package javaapplicationlistandsetcompdemo;
import java.util.*;
public class JavaApplicationListandSetCompDemo {

    static final int N=10000;
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        for(int i=0;i<N;i++)
            list.add(i);
        Collections.shuffle(list);
        Collection<Integer>myset = new HashSet<>(list);
        System.out.println("Time for Removing elements from set is:"+getRemoveTime(myset)+"milliseconds");
        Collection <Integer> mylist = new ArrayList<>(list);
        System.out.println("Time for Removing elements from list is:"+getRemoveTime(mylist)+"milliseconds");
    }

    private static long getRemoveTime(Collection<Integer> c) {
        // throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
        for(int i=0;i<N;i++)
            c.remove(i);
        long currentTime = System.currentTimeMillis();
        long startTime = 0;
        return currentTime-startTime;
    }
}
```

t - JavaApplicationListandSetCompDemo (run) X

```
run:
Time for Removing elements from set is:1586444626169milliseconds
Time for Removing elements from list is:1586444626233milliseconds
BUILD SUCCESSFUL (total time: 0 seconds)
```




SINGLETON AND UNMODIFIED COLLECTION:

- ❖ Collections class defines three constants
-EMPTY_SET,EMPTY_LIST AND EMPTY_MAP for empty set,an empty list and empty map.These Collections are immutable.
- ❖ The collection class also provides a method name singleton(object ob) for creating an immutable set containing only one item
- ❖ There are some unmodifiable methods defined by Collection class.



HASH TABLE

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. Hashtable is similar to HashMap except it is synchronized.

Hashtable class declaration

Let's see the declaration for java.util.Hashtable class.

1. **public class** Hashtable<K,V> **extends** Dictionary<K,V> **implements**
Map<K,V>

Constructor	Description
Hashtable()	It creates an empty hashtable having the initial default capacity and load factor.
Hashtable(int capacity)	It accepts an integer parameter and creates a hash table that contains a specified initial capacity.
Hashtable(int capacity, float loadFactor)	It is used to create a hash table having the specified initial capacity and loadFactor.
Hashtable(Map<? extends K,? extends V> t)	It creates a new hash table with the same mappings as the given Map.

Methods for HashTable

String toString()	It returns a string representation of the Hashtable object.
Collection values()	It returns a collection view of the values contained in the map.
boolean contains(Object value)	This method returns true if some value equal to the value exists within the hash table, else return false.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the hash table, else return false.
boolean containsKey(Object key)	This method return true if some key equal to the key exists within the hash tab else return false.
boolean isEmpty()	This method returns true if the hash table is empty; returns false if it contains least one key.
protected void rehash()	It is used to increase the size of the hash table and rehashes all of its keys.
V get(Object key)	This method returns the object that contains the value associated with the key.
V remove(Object key)	It is used to remove the key and its value. This method returns the value associated with the key.
int size()	This method returns the number of entries in the hash table.

```
package javaapplicationhashtableprog;
import java.util.*;
public class JavaApplicationHashTableProg {

    public static void main(String[] args) {

        // Creating a Hashtable
        Hashtable<Integer, String> h =
            new Hashtable<Integer, String>();

        // Adding Key and Value pairs to Hashtable
        h.put(10, "Chaitanya");
        h.put(20, "Ajeet");
        h.put(30, "Peter");
        h.put(40, "Ricky");
        h.put(50, "Mona");

        for (Map.Entry m:h.entrySet())
        {
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}
```

JavaApplicationHashTableProg (run) X

```
run:
10Chaitanya
20Ajeet
30Peter
40Ricky
50Mona
BUILD SUCCESSFUL (total time: 0 seconds)
```



HASH MAP

Java HashMap class implements the map interface by using a hash table. It inherits AbstractMap class and implements Map interface.

Points to remember

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

public class HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializable

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java HashMap class

Constructor	Description
HashMap()	It is used to construct a default HashMap.
HashMap(Map<? extends K,? extends V> m)	It is used to initialize the hash map by using the elements of the given Map object m.
HashMap(int capacity)	It is used to initialize the capacity of the hash map to the given integer value, capacity.
HashMap(int capacity, float loadFactor)	It is used to initialize both the capacity and load factor of the hash map by using arguments.

Methods for HashMap

Method	Description
void clear()	It is used to remove all of the mappings from this map.
boolean isEmpty()	It is used to return true if this map contains no key-value mappings.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.

<code>boolean containsValue(Object value)</code>	This method returns true if some value equal to the value exists within the map else return false.
<code>boolean containsKey(Object key)</code>	This method returns true if some key equal to the key exists within the map, else return false.
<code>boolean equals(Object o)</code>	It is used to compare the specified Object with the Map.
<code>void forEach(BiConsumer<? super K, ? super V> action)</code>	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
<code>V get(Object key)</code>	This method returns the object that contains the value associated with the key.
<code>V getOrDefault(Object key, V defaultValue)</code>	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
<code>boolean isEmpty()</code>	This method returns true if the map is empty; returns false if it contains at least one key.

```
package javaapplicationhashmap;

import java.util.HashMap;
import java.util.Map;

public class JavaApplicationHashMap {

    public static void main(String[] args) {
        HashMap<Integer,String> hm=new HashMap<Integer,String>();
        System.out.println("Initial list of elements: "+hm);
        hm.put(100,"Amit");
        hm.put(101,"Vijay");
        hm.put(102,"Rahul");

        System.out.println("After invoking put() method ");
        for(Map.Entry m:hm.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}
```

- JavaApplicationHashMap (run) ×

run:

Initial list of elements: {}

After invoking put() method

100 Amit

101 Vijay

102 Rahul

BUILD SUCCESSFUL (total time: 0 seconds)

DIFFERENCE BETWEEN HASHTABLE AND HASHMAP

HashMap	Hashtable
1) HashMap is non synchronized . It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized . It is thread-safe and can be shared with many threads.
2) HashMap allows one null key and multiple null values .	Hashtable doesn't allow any null key or value .
3) HashMap is a new class introduced in JDK 1.2 .	Hashtable is a legacy class .
4) HashMap is fast .	Hashtable is slow .
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is traversed by Iterator .	Hashtable is traversed by Enumerator and Iterator .
7) Iterator in HashMap is fail-fast .	Enumerator in Hashtable is not fail-fast .
8) HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.




TREEMAP

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- ❖ Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- ❖ Java TreeMap contains only unique elements.
- ❖ Java TreeMap cannot have a null key but can have multiple null values.
- ❖ Java TreeMap is non synchronized.
- ❖ Java TreeMap maintains ascending order.



```
public class TreeMap<K,V> extends AbstractMap<K,V> implements  
NavigableMap<K,V>
```

TreeMap class Parameters

Let's see the Parameters for java.util.TreeMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructor	Description
TreeMap()	It is used to construct an empty tree map that will be sorted using the natural order of key.
TreeMap(Comparator<? super K> comparator)	It is used to construct an empty tree-based map that will be sorted using the comparator comp.
TreeMap(Map<? extends K,? extends V> m)	It is used to initialize a treemap with the entries from m , which will be sorted using the natural order of the keys.
TreeMap(SortedMap<K,? extends V> m)	It is used to initialize a treemap with the entries from the SortedMap sm , which will be sorted in the same order as sm .

Methods For TreeMap

<code>boolean containsKey(Object key)</code>	It returns true if the map contains a mapping for the specified key.
<code>boolean containsValue(Object value)</code>	It returns true if the map maps one or more keys to the specified value.
<code>K firstKey()</code>	It is used to return the first (lowest) key currently in this sorted map.
<code>V get(Object key)</code>	It is used to return the value to which the map maps the specified key.
<code>K lastKey()</code>	It is used to return the last (highest) key currently in the sorted map.
<code>V remove(Object key)</code>	It removes the key-value pair of the specified key from the map.
<code>Set<Map.Entry<K,V>> entrySet()</code>	It returns a set view of the mappings contained in the map.
<code>int size()</code>	It returns the number of key-value pairs exists in the hashtable.
<code>Collection values()</code>	It returns a collection view of the values contained in the map.


```
package javaapplicationtreemap;

import java.util.Map;
import java.util.TreeMap;

public class JavaApplicationTreeMap {

    public static void main(String[] args) {
        TreeMap<Integer,String> map=new TreeMap<Integer,String>();
        map.put(100,"Amit");
        map.put(102,"Ravi");
        map.put(101,"Vijay");
        map.put(103,"Rahul");

        for(Map.Entry m:map.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}
```

- JavaApplicationTreeMap (run) ×

run:

100 Amit

101 Vijay

102 Ravi

103 Rahul

BUILD SUCCESSFUL (total time: 2 seconds)

Thank you!

