Collection framework

javatpoint

<https://www.javamex.com/tutorials/collections/how_to_choose_2.shtml>

Collection: group

Collection:Interface

Collections:Class

Framework:

* Semi complete application
* It represents a set of classes and interface

blue:inteface

Iterable

Collection

Green class

List

Duplicates are allowed

Natural order

Allow multiple null

Set

No Duplicates

Unorder

nnnn

N

Queue

FIFO,allow duplicate

Natural order

SortedSet

AbstractSet

Priority Queue,return from smallest value

ArrayList

Sequential memory storage.Fast searching.

Initial capacity 10.if full increase by half of existing

Dequeue

Can add and remove from both ends

Navigable set

HashSet

Unorder,can contain null,

Default size 16

TreeSet

Sorted ascending,no null

LinkedList

Dynamic memory Storage

Fast insertion and deletion.Searching is slow beacause it internally iterate the list..Internally it is doubly link list

ArrayDeQueue,

null not allow

Stack

Synchronized

LIFO

Vector

Synchronised

Same as ArrayList.Faster.

Initial capacity 10.if full increase by its existing

LinkedHashSet

ordered

*In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updation in object, thread must lock the particular segment in which thread want to operate.This type of locking mechanism is known as****Segment locking or bucket locking****.Hence at a time 16 updation operations can be performed by threads*

Unlike Hashtable and Synchronized Map, it never locks whole Map, instead, it divides the map into segments and locking is done on those.  
  
*ConcurrentHashMap is better than HashTable.*

Identity Hashmap.

Equal method not overrided.

Use to compare objects by reference

LinkedHashMap.

ordered

HashMap

Not Synchronised

may have one null key and multiple null values.

No Order

ConcurrentHashMap

ThreadSafe

ConcurrentMap

Properties

Key and value both are string.

The Properties class provides methods to get data from the properties file and store data into the properties file

HashTable

synchronised

Null not allowed.

objects used as keys must implement the hashCodemethod and the equals method.

AbstractMap

Sorted Map

Map

Key Value pair . Un-order

Navigable map

Red black tree based.TreeMap,sorted according to key,key can not be null,value can be null.

Weak hasmap ,when key is not referring to any object ,it is removed from weak hashmap

|  |  |  |
| --- | --- | --- |
| Enumeration | Iterator | ListIterator |
| Fail safe | Fail fast | Fail fast |
|  |  | For List collections only |
| Traverse in forward direction | Traverse in forward direction | Traverse in both direction |
| remove() absent | Remove()present | Add() and remove() methods |
| hasMoreElements(),  nextElement() | hasNext(),  next() | hasNext(),  next(),  previous().  hasPrevious() |
| It works on clone of collection |  |  |

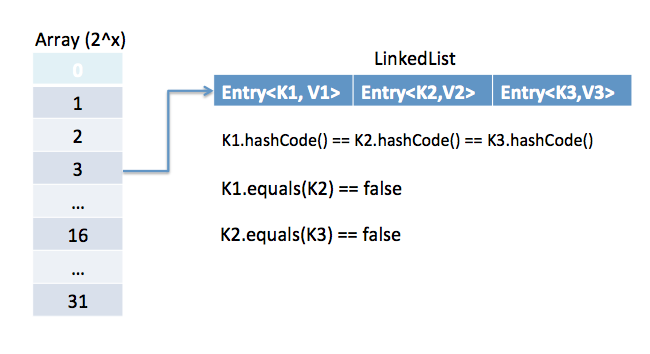
**Internal Working of HashMap:**

1. **How HashMap works in Java?**

HashMap stores key-value pair in Map.Entry static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.

When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there is an already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and stores this key-value Entry.

When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return its value. The below image will explain these detail clearly.



The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is **16** and load factor is 0.75. The threshold is capacity multiplied by load factor and whenever we try to add an entry if map size is greater than the threshold, HashMap rehashes the contents of the map into a new array with a larger capacity. The capacity is always the power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from the database, it’s a good idea to initialize the HashMap with correct capacity and load factor.

Equals and hascode contract:

If equals method return true for two objects then hascode method should return same int value for both objects

Two distinct objects can return same hascode.

|  |  |  |
| --- | --- | --- |
| Sr No | HashMAP | HashTable |
| 1 | Not Synchronised | Synchronised |
| 2 | may have one null key and multiple null values. | Null not allowed |
|  | fast | slow |
|  | Abstract class | Dictionary class |
|  | Use iterator | Iterator, enumeration |

|  |  |  |
| --- | --- | --- |
|  | collection | Internal implementation |
|  | ArrayList | Dynamic array |
|  | Vector | Dynamic array |
|  | LinkedList | Doubly Linked List |
|  | HashMap,HasTable | Hashing |
|  | TreeMAp,TreeSet | Red black tree |
|  | hashSET | Hashmap, |

put(K k, V v){

                return putV(hash(k), k, v, )

}

hash(K k)

                return k==null ? 0 : k.hascode()

boolean add(E e){

                return hashmap.put(1, PRESENT)==null;

}

boolean add(E e){

                return hashmap.put(2, PRESENT)==null;

}

boolean add(E e){

                return hashmap.put(1, PRESENT)==null;

}

# CopyOnWriteArrayList in java

* As the name indicates, CopyOnWriteArrayList creates a Cloned copy of underlying ArrayList, for every update operation at certain point both will be synchronized automatically ,which is taken care by JVM. Therefore there is no effect for threads which are performing read operation.
* It is costly to use because for every update operation a cloned copy will be created. Hence CopyOnWriteArrayList is the best choice if our frequent operation is read operation.
* It extends object and implements Serializable, Cloneable, Iterable<E>, Collection<E>, List<E> and RandomAccess
* The underlined data structure is grow-able array.
* It is thread-safe version of ArrayList.
* Insertion is preserved, duplicates are allowed and heterogeneous Objects are allowed.
* The main important point about CopyOnWriteArrayList is Iterator of CopyOnWriteArrayList can not perform remove operation otherwise we get Run-time exception saying UnsupportedOperationException.

WEAK HASHMAP

|  |
| --- |
| import java.util.\*;  class WeakHashMapDemo  {      public static void main(String args[])throws Exception      {          WeakHashMap m = new WeakHashMap();          Demo d = new Demo();            // puts an entry into WeakHashMap          m.put(d," Hi ");          System.out.println(m);            d = null;            // garbage collector is called          System.gc();            // thread sleeps for 4 sec          Thread.sleep(4000); .            System.out.println(m);      }  }    class Demo  {      public String toString()      {          return "demo";      }        // finalize method      public void finalize()      {          System.out.println("finalize method is called");      }  } |

Output:

{demo = Hi}

finalize method is called

{ }