|  |  |
| --- | --- |
| V Iterator | ListIterator |
| 1. Iterator is used for traversing List and Set both. | ListIterator to traverse List only, we cannot traverse Set using ListIterator. |
| 1. We can traverse in only forward direction using Iterator. | we can traverse a List in both the directions (forward and Backward). |
| 1. We cannot obtain indexes while using Iterator | We can obtain indexes at any point of time while traversing a list using ListIterator. |
| 1. We cannot add element to collection while traversing it using Iterator | We can add element at any point of time while traversing a list using ListIterator. |
| 1. We cannot replace the existing element value when using Iterator. | By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | List | Set | Map |
| Duplicate | Allows duplicate | Doesn’t allow duplicate elements | Its key –values. doesn’t allow duplicate keys and allow duplicate values |
| Null values | It allows any no of null values | Its allows single null values | Its allows single null key and no of values |
| order | Its ordered grouping elements | Its unordered grouping elements | Map doesn’t stores the elements in a order, but some of them like tree sort the map is ascending order |
| Commonly used classes | Arraylist,linkedlist,vector | Hashset,linkedhashset,treeset | Hashmap,hashtable,linkedhashmap,Treemap |

## When to use List, Set and Map in Java?

1) If you do not want to have duplicate values in the database then Set should be your first choice as all of its classes do not allow duplicates.  
2) If there is a need of frequent search operations based on the index values then List (ArrayList) is a better choice.  
3) If there is a need of maintaining the insertion order then also the List is a preferred collection interface.  
4) If the requirement is to have the key & value mappings in the database then Map is your best bet.

|  |  |
| --- | --- |
| ArrayList | Vector |
| 1.Its not Synchronized | Its synchronized |
| 2. its fast | Its slow |
| 3.its not legacy class  it is introduced in JDK 1.2. | Its legacy class |
| 4.it uses iterator | Its uses enumerator as well as iterator | |

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1.Its Dynamically array to store the elements | Its double linked list to store the elements |
| 2.its slow. it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Its fast |
| 3.its better for storing and accessing data | Its better for manipulating data |

|  |  |  |
| --- | --- | --- |
| Hashset | Treeset | LinkedHashset |
| 1.its doesn’t maintain any order of element | It will maintain in ascending order | It will miaintain insertion order |

|  |  |
| --- | --- |
| HashMap | HashTable |
| 1.Its non synchronized, It is not-thread safe | Its synchronized |
| 2.its fast | Its slow |
| 3.its allow single null key and multiple null values | Its doesn’t allow any null key or value |
| 4.hasmap I traverse by iterator | Hash table is traverse by enumerator and iterator |

Linkedhashmap

* A LinkedHashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is same as HashMap instead maintains insertion order.

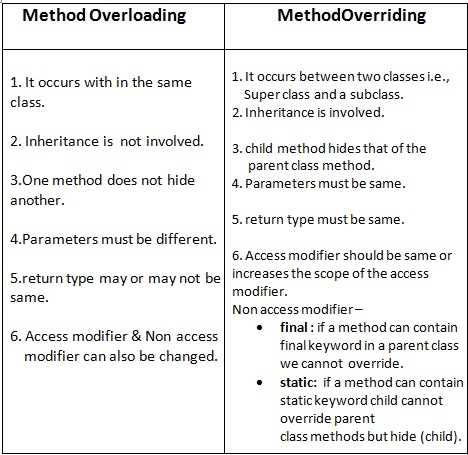
TreeMap

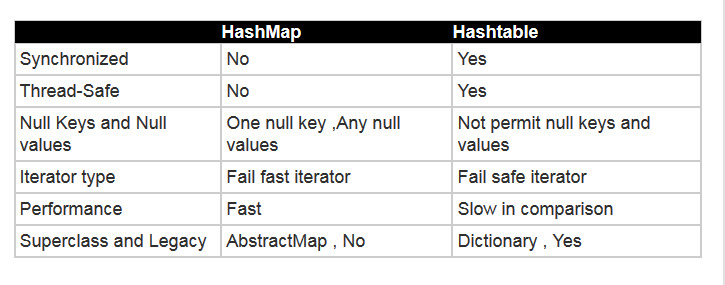
* A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* It contains only unique elements.
* It cannot have null key but can have multiple null values.
* It is same as HashMap instead maintains ascending order.

# Queue:  FIFO(First In First Out) manner.  first element is removed first and last element is removed at last.

|  |  |
| --- | --- |
| Comparable | Comparator |
| 1.it provide compare to() method to sort elements | Its provide Compare() method to sort the elements |
| 2.”Its single sorting sequence | 2.Its multiple sorting sequence |
| 3.it affect original class | It doesn’t affect original class |

|  |  |  |
| --- | --- | --- |
| **No.** | **Throw** | **throws** |
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| 2) | Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| 3) | Throw is followed by an instance. | Throws is followed by class. |
| 4) | Throw is used within the method. | Throws is used with the method signature. |
| 5) | You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |





|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc. |
| 2) Comparable **affects the original class** i.e. actual class is modified. | Comparator **doesn't affect the original class** i.e. actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is found in **java.lang** package. | Comparator is found in **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method. |

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) **Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

1. Equals() and hash code method:

2. If you are overriding equals method then you should override hashcode() also.

3. If two objects are equal then they must have same hashcode.

4. If two objects have same hashcode then they may or may not be equal

5. Always use same attributes to generate equals and hashcode as in our case we have used name.

Here are two rules that are good to know about implementing the hashCode() method in your own classes, if the hashtables in the Java Collections API are to work correctly:

1. If object1 and object2 are equal according to their equals() method, they must also have the same hash code.
2. If object1 and object2 have the same hash code, they do NOT have to be equal too.

In shorter words:

1. If equal, then same hash codes too.
2. Same hash codes no guarantee of being equal.

**LinkedList:**

**Java internally implements Double Linked List structure.**

It follows order of insertion.

LinkedList implements both List interface and also Queue interface.

**LinkedList is faster than ArrayList when it comes to insertions and deletions.**

**ArrayList is faster than LinkedList when it comes to just iterating the data and searching for data.**