Collection framework:

- A unified architecture for representing and manipulating collections.

- contains Interfaces, Implementations, algorithms.

* Collection : object that groups multiple elements into a single unit.
* Used to store, retrieve , manipulate and communicate aggregate data.
* Collection Interface : encapsulate different types of collections.
* It contains Set, List, Queue, Deque, SortedSet Interfaces.
* Super interface : Iterable. [ method: forEach()]
* Default methods :

1. Default Stream<E> parallelStream(): parallel stream with this collection as its source.
2. Default Boolean removeIf( Predicate< ? super E> filter) : removes all of the elements of this collection that satisfy the given predicate.
3. default Spliterator<E> spliterator()
4. default Stream<E> stream()

* Abstract methods:

1. boolean add( E e) :
2. boolean addAll(Collection <? Extends E> c )
3. void clear()
4. boolean contains(Object o)
5. boolean containsAll(Collection <?> c)
6. boolean equals (Object o)
7. int hashCode()
8. boolean isEmpty()
9. Iterator<E> iterator()
10. boolean remove (Object o)
11. boolean removeAll (Collection<?> c)
12. boolean retainAll( Collection<?> c)
13. int size()
14. Object[] toArray()
15. <T> T[] toArray(T[] a): Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

* Set Interface:
* No duplicate elements
* Only one null element
* It contains class HashSet, class LinkedHashSet, SortedSet Interface

1. HashSet:

* Allows null
* Iteration order is not constant
* Not synchronized
* It represents the collection that uses a hash table for storage.
* Hashing is used to store the elements in the HashSet.
* unique items.
* Constructors: HashSet(), HashSet(Collection<? Extends E> c), HashSet( int initicalCapacity), Hashset(Int iniCap, float loadFactor) [default loadFactor: 0.75]
* Methods: -

1. boolean add(E e)
2. void clear()
3. Object clone()
4. boolean contains(Object o)
5. boolean isEmpty()
6. Iterator<E> iterator()
7. boolean remove(Object o)
8. int size()
9. Spliterator<E> spliterator(): Creates a late-binding and fail-fast Spliterator over the elements in this set.
10. LinkedHashSet:

* Maintains doubly linkedlist
* Defines insertion order
* Contains unique elemnents and Permits null elements
* Not synchronized
* All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, Set<E>
* Constructors: LinkedHashSet(), LinkedHashSet(Collection <? Extends E> c), LinkedHashSet(int initialCapacity), LinkedHashSet(int iniCap, float loadFactor) [initialCapacity: 16, load factor: 0.75]
* Methods:

1. from hashset: add, clear, clone, contains, isEmpty, iterator, remove, size
2. Set: add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray,
3. Collection: parallelStream, removeIf, stream
4. Object: finalize, getClass, notify, notifyAll, wait, wait, wait
5. AbstractSet: equals, hashCode, removeAll
6. AbstractCollection: addAll, containsAll, retainAll, toArray, toArray, toString
7. Iterable: forEach
8. Spliterator<E> spliterator():Creates a late-binding and fail-fast Spliterator over the elements in this set.
9. SortedSet Interface:

* Provides a total ordering to its element
* Natural ordereing ( ascending order)
* Methods:

1. Comparator<? Super E> comparator()
2. E first()
3. SortedSet<E> headset(E toElement): set whose elements are strictly less than toElement.
4. E last()
5. Default Spliterator<E> spliterator()
6. SortedSet<E> subset( E fromElement, E toElement)
7. SortedSet<E> tailSet( E fromElement)
8. Methods from Set, Collection and Iterable.

D: TreeSet Class:

* Implements SortedSet
* All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, NavigableSet<E>, Set<E>, SortedSet<E>
* Not Synchronized
* Unique elements
* Ascending order
* Constructors: TreeSet(), TreeSet(Collection< ? extends E> c), TreeSet(Comparator< ?super E> comparator), TreeSet(SortedSet<E> s)
* Methods:

1. boolean add( E e)
2. boolean addAll( Collection< ? extends E> c)
3. E ceiling(E e): Returns the least element in this set greater than or equal to the given element
4. void clear()
5. Object clone()
6. Comparator< ? super E> comparator()
7. boolean contains(Objct o)
8. Iterator<E> descendingIterator()
9. NavigableSet<E> descendingSet()
10. E first()
11. E floor(E e)
12. SortedSet<E> headset(E toElement)
13. NavigableSet<E> headset(E toElemnt, bollean inclusive)
14. E higher( E e)
15. boolean isEmpty()
16. Iterator<E> iterator()
17. E last()
18. E lower(E e)
19. E pollFirst()
20. E pollLast()
21. boolean remove(Object o)
22. int size()
23. Spliterator<E> spliterator()
24. NavigableSet<E> subset(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)
25. SortedSet<E> subset(E fromElement, E toElement)
26. SortedSet<E> tailSet(E fromElement)
27. NavigableSet<E> tailSet( E fromElement, boolean inclusive)

* List Interface:
* Ordered/ sequence collection
* Duplicate elements and allows null values
* Element can be accessed by index
* Positional access: methods: E get(int index), E set(int index, E element), add, addAll, remove
* Search : int indexof(), int lastIndexOf()
* Iteration : ListIterator() -bidirectional access
* Range views : List<E> subList(int fromIndex, int toIndex)) – performs range operations.
* default void sort(Comparator< ? super E> c)
* classes and methods in list

1. Class ArrayList:

* Resizable array/ dynamic array
* Duplicate allowed
* Element with different data types is allowed
* Maintains insertion order
* Random access possible
* Null is allowed
* Unsynchronized
* Equivalent to vector
* All Implemented Interfaces:Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess
* Constructors: ArrayList(), ArrayList(Collection< ? extends E> c), ArrayList( int initialCapacity)
* Methods :

1. All list interface methods and Iterator, ListIterator methods
2. void ensureCapacity(int minCapacity)
3. void forEach(Consumer< ? super E> action)
4. ListIterator<E> listIterator()
5. ListIterator<E> listIterator(int index) – [ I is starting position]
6. removeIf(Predicate<? Super E> filter)
7. protected void removeRange(int fromIndex, int toIndex)
8. void replaceAll(UnaryOperator<E> operator)
9. boolean retainAll(Collection<?> c)
10. List<E> subList(int fromIndex, int toIndex
11. Object[] toArray()
12. <T> T[] toArray(T[] a)
13. void trimToSize()
14. Class LinkedList:

* Doubly-linked list implementation of the List and Deque interfaces
* Permits all elements [ null/duplicate]
* Not synchronized
* Maintains the insertion order
* Manipulation is fast no shifting is required.
* All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, List<E>, Queue<E>
* Constructors: LinkedList(), LinkedList(Collection <? Extends E> c)
* Methods:

1. Includes Methods of list, queue , dqueue interface, Iterator
2. Vector Class:

* Growable/shrink array of objects.
* Synchronized
* All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess
* Constructor: Vector(), Vector( Collection< ? extends E> c), Vector( int initialCapacity), Vector( int initialCapacity, int capacityIncrement)
* Methods:

1. Inherited methods
2. int capacity()
3. void copyInto(Object[] anArray)
4. E elementAt( int index)
5. Enumeration<E> elements()
6. Stack class;

* represents a last-in-first-out (LIFO) stack of objects
* child of Vector class
* All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess
* Constructor: Stack()
* Methods:

1. boolean empty()
2. E peek()
3. E pop()
4. E push(E item)
5. Int search(Object o)
6. All inherited methods from vector and some from Object, Collection

* Queue: FIFO
* provide additional insertion, extraction, and inspection operations
* Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either null or false, depending on the operation).

|  |  |  |
| --- | --- | --- |
|  | Throws exception | Returns special value |
| Insert | [add(e)](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html#add-E-)- unchecked exp | [offer(e)](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html#offer-E-) - false |
| Remove | [remove()](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html#remove--)-NoSuchElementException | [poll()](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html#poll--)- null |
| Examine | [element()](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html#element--)- NoSuchElementException | [peek()](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html#peek--)- head of queue/null |

* Priority Queue: order of elements according to a supplied comparator/ elements natural order
* Null elements not allowed
* do not define element-based versions of methods equals() and hashCode() but instead inherit the identity based versions from class Object
* add() Exceptions: **Throws:**

1. IllegalStateException - if the element cannot be added at this time due to capacity restrictions
2. ClassCastException - if the class of the specified element prevents it from being added to this queue
3. NullPointerException - if the specified element is null and this queue does not permit null elements
4. IllegalArgumentException - if some property of this element prevents it from being added to this queue

* offer(): Exception Throws:

1. ClassCastException - if the class of the specified element prevents it from being added to this queue
2. NullPointerException - if the specified element is null and this queue does not permit null elements
3. IllegalArgumentException - if some property of this element prevents it from being added to this queue

* Child Class/ Interfaces of Queue:

1. Class PriorityQueue:

* Based on a priority heap.
* Unbounded but has an internal capacity governing the size of an array used to store the elements on the queue.
* Grow automatically
* It holds the elements or objects which are to be processed by their priorities.
* Not synchronized
* Ordered according natural ordering/ by comparator
* Not permit null
* All Implemented Interfaces: Serializable, Iterable<E>, Collection<E>, Queue<E>
* Not permit insertion of non-comparable objects
* The head of this queue is the least element with respect to the specified ordering. If multiple elements are tied for least value, the head is one of those elements -- ties are broken arbitrarily. The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
* Constructors: PriorityQueue()- [initial capacity- 11, natural order], PriorityQueue(Collection< ? extends E> c), PriorityQueue(Comparator<? super E> comparator), PriorityQueue(int initialCapacity), PriorityQueue(int initialCapacity, Comparator<? super E> comparator), PriorityQueue(PriorityQueue<? extends E> c), PriorityQueue(SortedSet<? extends E> c)
* Methods:

1. boolean add(E e)
2. void clear()
3. Comparator<? super E> comparator()
4. boolean contains(Object o)
5. Iterator<E> iterator()
6. boolean offer(E e)
7. E peek()
8. E poll()
9. boolean remove(Object o)
10. int size()
11. Spliterator<E> spliterator()
12. Object[] toArray()
13. <T> T[] toArray(T[] a)
14. Other methods inherited from AbstractQueue, AbstractCollection, Object, Collection, Iterable.
15. Deque Iterface:

* A linear collection that supports element insertion and removal at both ends
* Double ended queue
* All Superinterfaces: Collection<E>, Iterable<E>, Queue<E>
* All Known Subinterfaces: BlockingDeque<E>
* All Known Implementing Classes: ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque, LinkedList
* Includes Method inherited from parent.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Summary of Deque methods | | | | |
|  | **First Element (Head)** | | **Last Element (Tail)** | |
|  | *Throws exception* | *Special value* | *Throws exception* | *Special value* |
| **Insert** | [addFirst(e)](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#addFirst-E-) | [offerFirst(e)](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#offerFirst-E-) | [addLast(e)](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#addLast-E-) | [offerLast(e)](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#offerLast-E-) |
| **Remove** | [removeFirst()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#removeFirst--) | [pollFirst()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#pollFirst--) | [removeLast()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#removeLast--) | [pollLast()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#pollLast--) |
| **Examine** | [getFirst()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#getFirst--) | [peekFirst()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#peekFirst--) | [getLast()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#getLast--) | [peekLast()](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html#peekLast--) |

1. ArrayDeque Class:

* Resizable-array implementation of the Deque interface
* Not thread safe
* Null not allowed
* All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, Queue<E>
* Constructors: ArrayDeque(), ArrayDeque( Collection<? Extends E> c), ArrayDeque(int numElements)
* Methods:
* Map:
* An object that maps keys to values
* Can not contain duplicate keys & each key can map to at most one value
* The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings.
* All Known Subinterfaces: Bindings, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>, LogicalMessageContext, MessageContext, NavigableMap<K,V>, SOAPMessageContext, SortedMap<K,V>
* All Known Implementing Classes: AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap
* Methods in map:

1. void clear()
2. default V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
3. default V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)
4. default V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
5. boolean containsKey(Object key)
6. boolean containsValue(Object value)
7. Set<Map.Entry<K,V>> entrySet()
8. boolean equals(Object o)
9. default void forEach(BiConsumer<? super K,? super V> action)
10. V get(Object key)
11. default V getOrDefault(Object key, V defaultValue)
12. int hashCode()
13. boolean isEmpty()
14. Set<K> keySet()
15. default V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)
16. V put(K key, V value)
17. void putAll(Map<? extends K,? extends V> m)
18. default V putIfAbsent(K key, V value)
19. V remove(Object key)
20. default boolean remove(Object key, Object value)
21. default V replace(K key, V value)
22. default boolean replace(K key, V oldValue, V newValue)
23. default void replaceAll(BiFunction<? super K,? super V,? extends V> function)
24. int size()
25. Collection<V> values()

* Map contains following Interface and class:

1. SortedMap Interface:

* A Map that further provides a total ordering on its keys. The map is ordered according to the natural ordering of its keys, or by a Comparator typically provided at sorted map creation time.
* All keys inserted into a sorted map must implement the Comparable interface (or be accepted by the specified comparator).
* All Known Subinterfaces: ConcurrentNavigableMap<K,V>, NavigableMap<K,V>
* All Known Implementing Classes: ConcurrentSkipListMap, TreeMap
* constructors for all sorted map implementations are:

1. A void (no arguments) constructor, which creates an empty sorted map sorted according to the natural ordering of its keys.
2. A constructor with a single argument of type Comparator, which creates an empty sorted map sorted according to the specified comparator.
3. A constructor with a single argument of type Map, which creates a new map with the same key-value mappings as its argument, sorted according to the keys' natural ordering.
4. A constructor with a single argument of type SortedMap, which creates a new sorted map with the same key-value mappings and the same ordering as the input sorted map.

* Methods in sorted map:

1. Comparator<? super K> comparator()
2. Set<Map.Entry<K,V>> entrySet()
3. K firstKey()
4. SortedMap<K,V> headMap(K toKey)
5. Set<K> keySet()
6. K lastKey()
7. SortedMap<K,V> subMap(K fromKey, K toKey)
8. SortedMap<K,V> tailMap(K fromKey)
9. Collection<V> values()
10. Methods inherited from Map.
11. TreeMap class:

* The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.
* All Implemented Interfaces: Serializable, Cloneable, Map<K,V>, NavigableMap<K,V>, SortedMap<K,V>
* Implements NavigableMap interface and extends AbstractMap class
* Can not contains null keys but can have null values, and unique elements
* Maintains ascending order
* Not synchronized
* Constructors: TreeMap(), TreeMap(Comparator<? super K> comparator), TreeMap(Map<? extends K,? extends V> m), TreeMap(SortedMap<K,? extends V> m)
* Methods :

1. Map.Entry<K,V> ceilingEntry(K key)
2. K ceilingKey(K key)
3. void clear()
4. Object clone()
5. Comparator<? super K> comparator()
6. boolean containsKey(Object key)
7. boolean containsValue(Object value)
8. NavigableSet<K> descendingKeySet()
9. NavigableMap<K,V> descendingMap()
10. Set<Map.Entry<K,V>> entrySet()
11. Map.Entry<K,V> firstEntry()
12. K firstKey()
13. Map.Entry<K,V> floorEntry(K key)
14. K floorKey(K key)
15. void forEach(BiConsumer<? super K,? super V> action)
16. V get(Object key)
17. SortedMap<K,V> headMap(K toKey)
18. NavigableMap<K,V> headMap(K toKey, boolean inclusive)
19. Map.Entry<K,V> higherEntry(K key)
20. K higherKey(K key)
21. Set<K> keySet()
22. Map.Entry<K,V> lastEntry()
23. K lastKey()
24. Map.Entry<K,V> lowerEntry(K key)
25. K lowerKey(K key)
26. NavigableSet<K> navigableKeySet()
27. Map.Entry<K,V> pollFirstEntry()
28. Map.Entry<K,V> pollLastEntry()
29. V put(K key, V value)
30. void putAll(Map<? extends K,? extends V> map)
31. V remove(Object key)
32. V replace(K key, V value)
33. boolean replace(K key, V oldValue, V newValue)
34. void replaceAll(BiFunction<? super K,? super V,? extends V> function)
35. int size()
36. NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)
37. SortedMap<K,V> subMap(K fromKey, K toKey)
38. SortedMap<K,V> tailMap(K fromKey)
39. NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)
40. Collection<V> values()
41. HashMap class:

* Hash table based implementation of the Map interface
* Permits null values and only one null key
* Unsynchronized
* No guarantees as to the order of the map
* Unique keys and If you try to insert the duplicate key, it will replace the element of the corresponding key.
* An instance of HashMap has two parameters that affect its performance: initial capacity[capacity at the time the hash table is created] and load factor[measure of how full the hash table is allowed to get before its capacity is automatically increased] default-.75.
* All Implemented Interfaces: Serializable, Cloneable, Map<K,V>
* Direct Known Subclasses: LinkedHashMap, PrinterStateReasons
* Constructors: HashMap(),HashMap(int initialCapacity), HashMap(int initialCapacity, float loadFactor), HashMap(Map<? extends K,? extends V> m)
* Methods :

1. void clear()
2. Object clone()
3. V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
4. V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)
5. V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
6. boolean containsKey(Object key)
7. boolean containsValue(Object value)
8. Set<Map.Entry<K,V>> entrySet()
9. void forEach(BiConsumer<? super K,? super V> action)
10. V get(Object key)
11. V getOrDefault(Object key, V defaultValue)
12. boolean isEmpty()
13. Set<K> keySet()
14. V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)
15. V put(K key, V value)
16. void putAll(Map<? extends K,? extends V> m)
17. V putIfAbsent(K key, V value)
18. V remove(Object key)
19. boolean remove(Object key, Object value)
20. V replace(K key, V value)
21. boolean replace(K key, V oldValue, V newValue)
22. boolean replace(K key, V oldValue, V newValue)
23. int size()
24. Collection<V> values()
25. AbstractMap: [equals](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html#equals-java.lang.Object-), [hashCode](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html" \l "hashCode--), [toString](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html" \l "toString--)
26. Object: [finalize](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#finalize--), [getClass](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html" \l "getClass--), [notify](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#notify--), [notifyAll](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html" \l "notifyAll--), [wait](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#wait--), [wait](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#wait-long-), [wait](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#wait-long-int-)
27. Map: [equals](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#equals-java.lang.Object-), [hashCode](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html" \l "hashCode--)
28. LinkedHashMap class:

* it maintains a doubly-linked list running through all of its entries.
* This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order).
* Permits null elements and only one null key
* Not synchronized.
* A linked hash map has two parameters that affect its performance: initial capacity and load factor.
* All Implemented Interfaces: Serializable, Cloneable, Map<K,V>
* Inherits hashMap and implements map interface
* Constructors: LinkedHashMap(), LinkedHashMap(int capacity), LinkedHashMap(int capacity, float loadFactor), LinkedHashMap(int capacity, float loadFactor, boolean accessOrder), LinkedHashMap(Map<? extends K,? extends V> m)
* Methods:

1. void clear()
2. boolean containsValue(Object value)
3. Set<Map.Entry<K,V>> entrySet()
4. void forEach(BiConsumer<? super K,? super V> action)
5. V get(Object key)
6. V getOrDefault(Object key, V defaultValue)
7. Set<K> keySet()
8. protected boolean removeEldestEntry(Map.Entry<K,V> eldest)
9. void replaceAll(BiFunction<? super K,? super V,? extends V> function)
10. Collection<V> values()
11. HashMap: [clone](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#clone--), [compute](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#compute-K-java.util.function.BiFunction-), [computeIfAbsent](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#computeIfAbsent-K-java.util.function.Function-), [computeIfPresent](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#computeIfPresent-K-java.util.function.BiFunction-), [containsKey](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#containsKey-java.lang.Object-), [isEmpty](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#isEmpty--), [merge](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#merge-K-V-java.util.function.BiFunction-), [put](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#put-K-V-), [putAll](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#putAll-java.util.Map-), [putIfAbsent](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#putIfAbsent-K-V-), [remove](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#remove-java.lang.Object-), [remove](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#remove-java.lang.Object-java.lang.Object-), [replace](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#replace-K-V-), [replace](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#replace-K-V-V-), [size](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#size--)
12. AbstractMap: [equals](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html#equals-java.lang.Object-), [hashCode](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html" \l "hashCode--), [toString](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html" \l "toString--)
13. Object: [finalize](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#finalize--), [getClass](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html" \l "getClass--), [notify](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#notify--), [notifyAll](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html" \l "notifyAll--), [wait](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#wait--), [wait](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#wait-long-), [wait](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#wait-long-int-)
14. Map: [compute](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#compute-K-java.util.function.BiFunction-), [computeIfAbsent](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#computeIfAbsent-K-java.util.function.Function-), [computeIfPresent](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#computeIfPresent-K-java.util.function.BiFunction-), [containsKey](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#containsKey-java.lang.Object-), [equals](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#equals-java.lang.Object-), [hashCode](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#hashCode--), [isEmpty](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#isEmpty--), [merge](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#merge-K-V-java.util.function.BiFunction-), [put](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#put-K-V-), [putAll](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#putAll-java.util.Map-), [putIfAbsent](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#putIfAbsent-K-V-), [remove](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#remove-java.lang.Object-), [remove](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#remove-java.lang.Object-java.lang.Object-), [replace](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#replace-K-V-), [replace](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#replace-K-V-V-), [size](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html#size--)

**Difference Summary:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Interfaces** |  |  |  |
| **Property** | **Collection** | **Set** | **SortedSet** | **List** | **Map** | **SortedMap** |
| Duplicates? | unspecified | no | no | yes | no | no |
| Ordering | unspecified | unspecified | logical | insertion | unspecified | logical |
| Modifiable? | optional | optional | optional | optional | optional | optional |
| Access | positional | positional | positional | positional | associative | associative |

