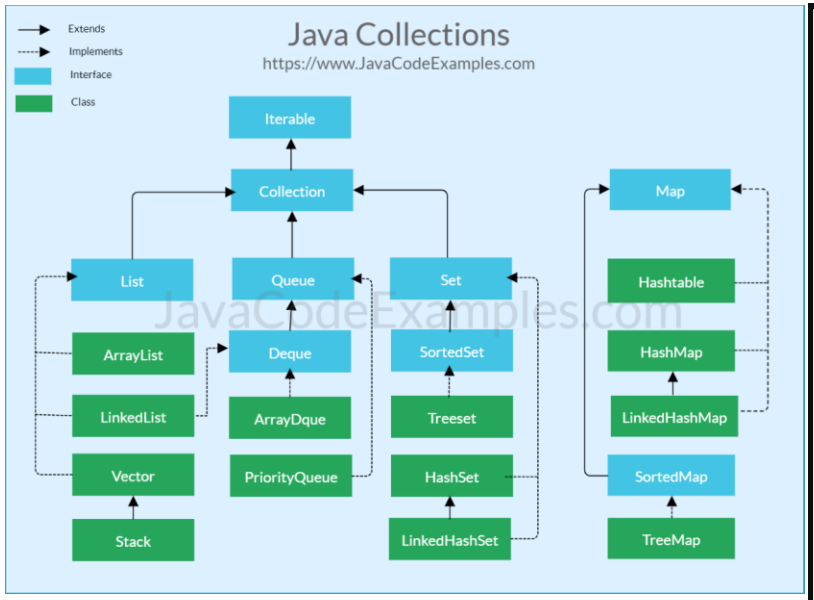
Collections

05506004

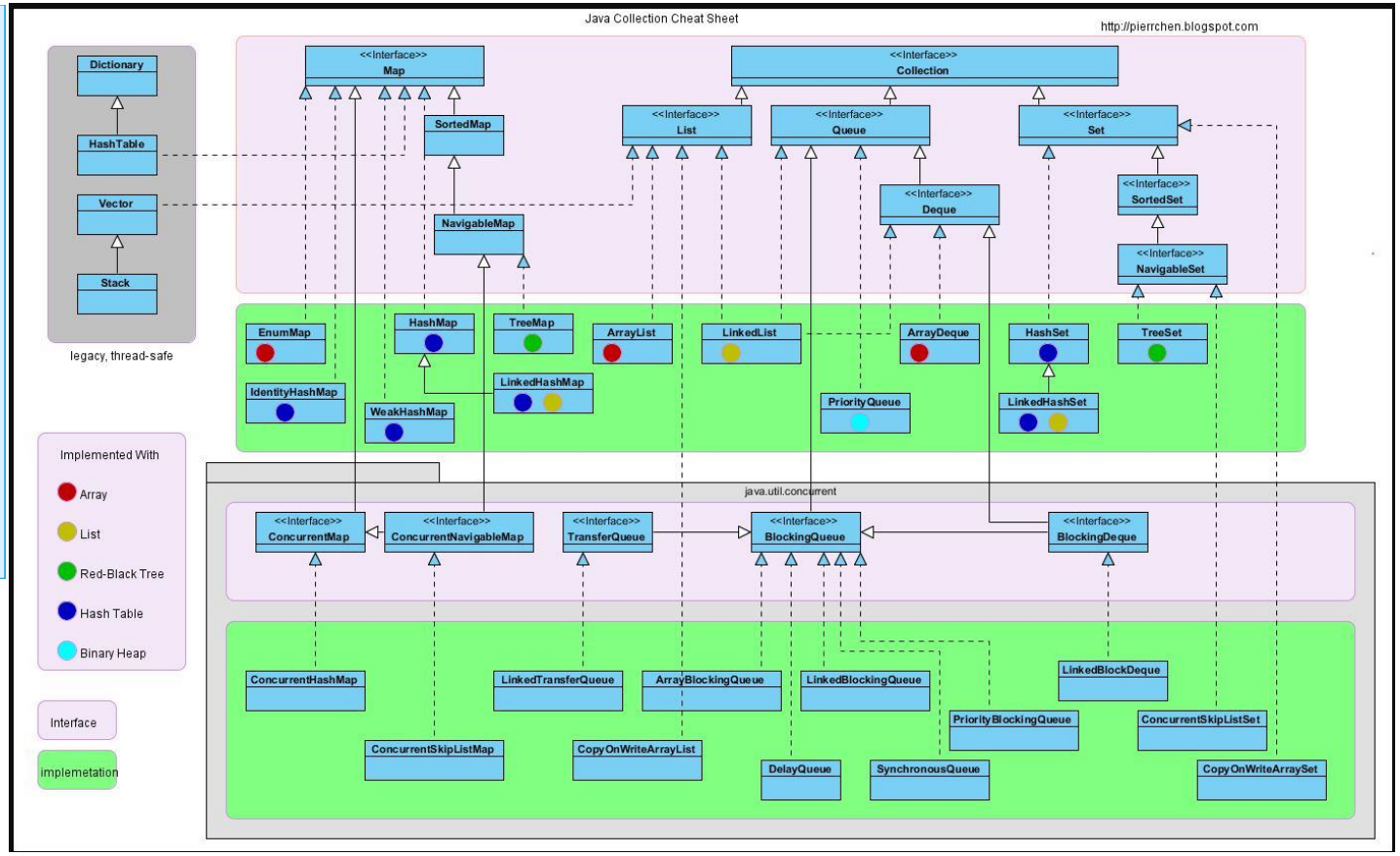
Outlines

- ArrayList, HashSet, PriorityQueue, HashMap
- Collection Interface
- Iterable Interface
- ArrayList
- HashSet
- PriorityQueue
- HashMap

Java Collections

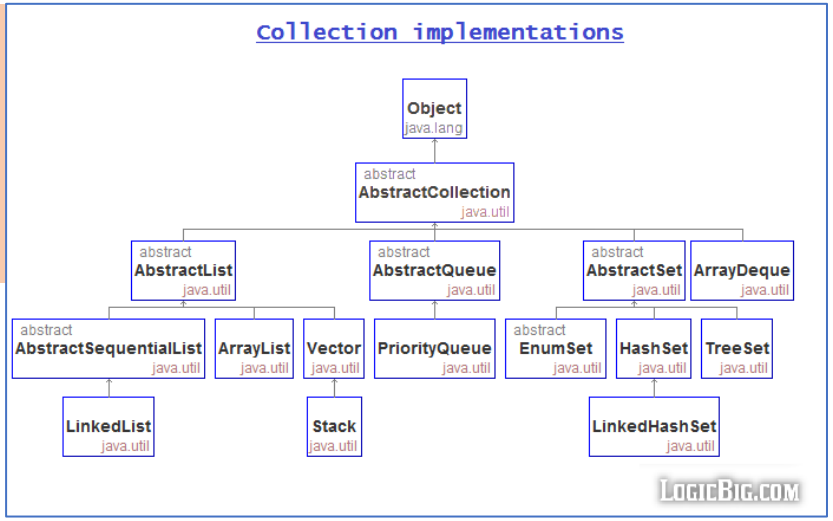


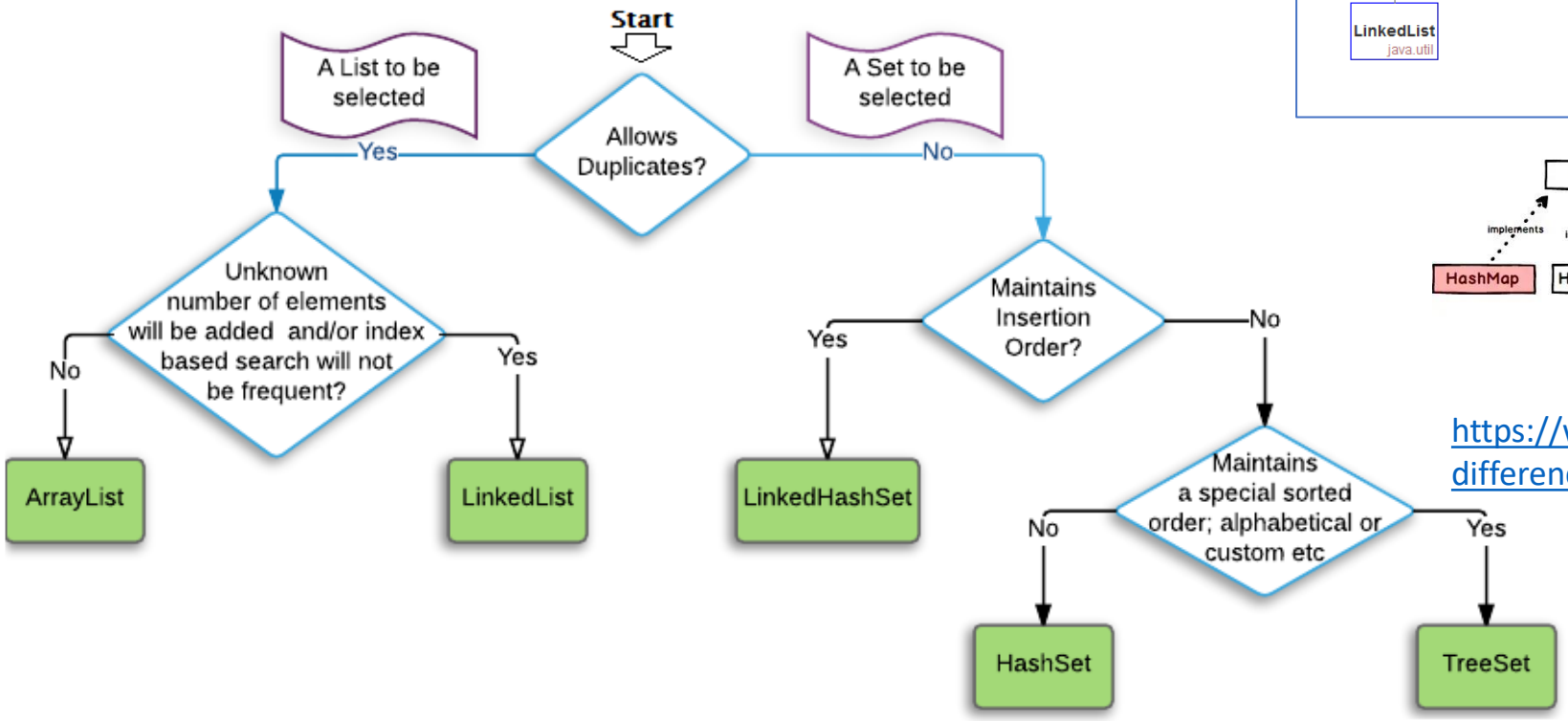
https://www.javacodeexamples.com/java-collection-framework-tutorial-with-examples/1641

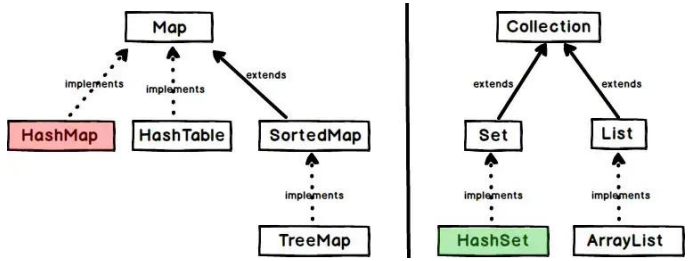


https://helpezee.wordpress.com/2019/08/15/java-collections-framework-cheat-sheet/

Different Characteristics







https://waytolearnx.com/2018/11/difference-entre-hashset-et-hashmap.html

https://www.logicbig.com/tutorials/core-java-tutorial/java-collections/java-collection-cheatsheet.html

Different Characteristics

	Duplicates Allowed	Elements Ordered	Elements Sorted	Synchronized
ArrayList	YES	YES	NO	NO
LinkedList	YES	YES	NO	NO
Vector	YES	YES	NO	YES
HashSet	NO	NO	NO	NO
LinkedHashSet	NO	YES	NO	NO
TreeSet	NO	YES	YES	NO
HashMap	NO	NO	NO	NO
LinkedHashMap	NO	YES	NO	NO
HashTable	NO	NO	NO	YES
TreeMap	NO	YES	YES	NO

https://www.linkedin.com/pulse/java-collections-table-cheat-sheet-apala-sengupta

Collections Implementations

Impl	ADT / DataStructure	Operations
ArrayList	List / Array of objects	add(E element), remove(int idx), get(int idx), etc
LinkedList	List, Deque / Doubly-linked list	get(int idx), remove(int idx), add(E elem), etc
Vector		Similar to ArrayList but slower because of synchronization.
Stack		Similar to Vector / ArrayList but slower because of synchronization.
HashSet	Set	add, remove, contains, size, **iteration**
LInkedHashSet	Set	add, remove, contains, size, itteration slightly slow that of HashSet, due to maintaining the linked list.
TreeSet	NavigableSet	add, remove, contains, iteration slower than HashSet.
EnumSet	Set	
PriorityQueue	Queue / Binary Heap	offer, poll, remove() and add, remove(Object), contains(Object), peek, element, and size
ArrayDeque	Dequeue / Resizable-array	remove, removeFirstOccurrence, removeLastOccurrence, contains, iterator.remove(), etc

https://www.logicbig.com/tutorials/core-java-tutorial/java-collections/java-collection-cheatsheet.html

Performance Characteristics

List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List
CopyOnWriteArrayList	O(n)	O(n)	O(1)	O(n)	O(1)	Array

Set	Add	Remove	Contains	Next	Size	Data Structure
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
EnumSet	O(1)	O(1)	O(1)	O(1)	O(1)	Bit Vector
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Red-black tree
CopyOnWriteArraySet	O(n)	O(n)	O(n)	O(1)	O(1)	Array
ConcurrentSkipListSet	O(log n)	O(log n)	O(log n)	O(1)	O(n)	Skip List

Queue	Offer	Peak	Poll	Remove	Size	Data Structure
PriorityQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedList	O(1)	O(1)	O(1)	O(1)	O(1)	Array
ArrayDequeue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)	O(n)	Linked List
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Array
PrioriritыBlockingQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
SynchronousQueue	O(1)	O(1)	O(1)	O(n)	O(1)	None!
DelayQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List

Map	Get	ContainsKey	Next	Data Structure
HashMap	O(1)	O(1)	O(h / n)	Hash Table
LinkedHashMap	O(1)	O(1)	O(1)	Hash Table + Linked List
IdentityHashMap	O(1)	O(1)	O(h / n)	Array
WeakHashMap	O(1)	O(1)	O(h / n)	Hash Table
EnumMap	O(1)	O(1)	O(1)	Array
TreeMap	O(log n)	O(log n)	O(log n)	Red-black tree
ConcurrentHashMap	O(1)	O(1)	O(h / n)	Hash Tables
ConcurrentSkipListMap	O(log n)	O(log n)	O(1)	Skip List

https://www.linkedin.com/pulse/big-o-java-collections-kunal-saxena

Collection<E> Interface

```
// Interface java.util.Collection<E>
// Basic Operations
abstract int size()                          // Returns the number of elements
abstract boolean isEmpty()                   // Returns true if there is no element

// "Individual Element" Operations
abstract boolean add(E element)              // Add the given element
abstract boolean remove(Object element)      // Removes the given element, if present
abstract boolean contains(Object element)    // Returns true if this Collection contains the given element

// "Bulk" (mutable) Operations
abstract void clear()                                // Removes all the elements
abstract boolean addAll(Collection<? extends E> c)   // Another Collection of E or E's subtypes
abstract boolean containsAll(Collection<?> c)        // Another Collection of any types
abstract boolean removeAll(Collection<?> c)
abstract boolean retainAll(Collection<?> c)

// Comparison - Objects that are equal shall have the same hashCode
abstract boolean equals(Object o)
abstract int hashCode()

// Array Operations
abstract Object[] toArray()       // Convert to an Object array
abstract <T> T[] toArray(T[] a)   // Convert to an array of the given type T
```

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

Iterator<E> interface

The super-interface `Iterable<E>` defines a mechanism to iterate (or traverse) through all the elements of a `Collection<E>` object via a so-called `Iterator<E>` object. The `Iterable<E>` interface declares one `abstract` method to retrieve the `Iterator<E>` object associated with the `Collection<E>`.

```
// Interface java.lang.Iterable<E>
abstract Iterator<E> iterator();
   // Returns the associated Iterator instance that can be used to traverse thru all the elements
```

The `Iterator<E>` interface declares the following `abstract` methods for traversing through the `Collection<E>`.

```
// Interface java.util.Iterator<E>
abstract boolean hasNext()  // Returns true if it has more elements
abstract E next()           // Returns the next element (of the actual type)
```

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

Why we use iterator interface in Java?

Java Iterator Interface of java collections **allows us to access elements of the collection and is used to iterate over the elements in the collection(Map, List or Set)**. It helps to easily retrieve the elements of a collection and perform operations on each element.

Nov 9, 2563 BE

geeksforgeeks.org
https://www.geeksforgeeks.org › iterator-interface-in-java

```
List<String> lst = new ArrayList<>();   // JDK 7 type inference
lst.add("alpha");
lst.add("beta");
lst.add("charlie");

// (1) Using the associated Iterator<E> to traverse through all elements
// Retrieve the Iterator associated with this List via the iterator() method
Iterator<String> iter = lst.iterator();
// Transverse thru this List via the Iterator
while (iter.hasNext()) {
   // Retrieve each element and process
   String str = iter.next();
   System.out.println(str);
}
```

iterator vs. for-each

You might need to use `iterators` if you need to modify collection in your loop. First approach will throw exception.

```
for (String i : list) {
    System.out.println(i);
    list.remove(i); // throws exception
}

Iterator it=list.iterator();
while (it.hasNext()){
    System.out.println(it.next());
    it.remove(); // valid here
}
```

https://stackoverflow.com/questions/18508786/for-each-vs-iterator-which-will-be-the-better-option

java.util.List<E>

```
// Interface java.util.List<E>
// Operations at a specified index position
abstract void add(int index, E element)     // add at index
abstract E set(int index, E element)        // replace at index
abstract E get(int index)                   // retrieve at index without remove
abstract E remove(int index)                // remove at index
abstract int indexOf(Object obj)
abstract int lastIndexOf(Object obj)

// Operations on a range fromIndex (inclusive) toIndex (exclusive)
abstract List<E> subList(int fromIndex, int toIndex)
```

```
// Utility Class java.util.Collections
static <T> int binarySearch(List<? extends T> lst, T key, Comparator<? super T> comp)
static <T> int binarySearch(List<? extends Comparable<? super T>> lst, T key)
   // Searches for the specified key using binary search

static <T> void sort(List<T> lst, Comparator<? super T> comp)
static <T extends Comparable<? super T>> void sort(List<T> lst)

static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)
static <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll)
static <T> T min(Collection<? extends T> coll, Comparator<? super T> comp)
static <T extends Object & Comparable<? super T>> T min(Collection<? extends T> coll)
```

The utility class `java.util.Collections` provides many useful algorithms for collection. Some work for any `Collections`; while many work for `Lists` (with numerical index) only.

Mutating Operators

```
// Utility Class java.util.Collections
static void swap(List<?> lst, int i, int j)   // Swaps the elements at the specified indexes
static void shuffle(List<?> lst)              // Randomly permutes the List
static void shuffle(List<?> lst, Ramdon rnd)  // Randomly permutes the List using the specified source or randomness
static void rotate(List<?> lst, int distnace) // Rotates the elements by the specified distance
static void reverse(List<?> lst)              // Reverses the order of elements
static <T> void fill(List<? super T>, T obj)  // Replaces all elements with the specified object
static <T> void copy(List<? super T> dest, List<? extends T> src)  // Copies all elements from src to dest
static <T> boolean replaceAll(List<T> lst, T oldVal, T newVal)     // Replaces all occurrences
```

[TODO] example

Sub-List (Range-View) Operations

The `List<E>` supports range-view operation via `.subList()` as follows. The returned `List` is backup by the given `List`, so change in the returned `List` are reflected in the original `List`.

```
// Interface java.util.List<E>
List<E> subList(int fromIdx, int toIdx)
```

The Utility class `Collections` supports these sub-list operations:

```
// Utility Class java.util.Collections
static int indexOfSubList(List<?> src, List<?> target)
static int lastIndexOfSubList(List<?> src, List<?> target)
```

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

Sorting (revisited)

```java
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class StringPrimitiveComparatorJ8Test {  // JDK 8
   public static void main(String[] args) {
      // Use a customized Comparator for Strings
      Comparator<String> strComp = (s1, s2) -> s1.compareToIgnoreCase(s2);
         // The lambda expression create an instance of an anonymous inner class implements
         //   Comparator<String> with the body of the single-abstract-method compare()

      // Sort and search an "array" of Strings
      String[] array = {"Hello", "Hi", "HI", "hello", "Hello"};  // with duplicate
      Arrays.sort(array, strComp);
      System.out.println(Arrays.toString(array));  //[Hello, hello, Hello, Hi, HI]
      System.out.println(Arrays.binarySearch(array, "Hello", strComp)); //2
      System.out.println(Arrays.binarySearch(array, "HELLO", strComp)); //2 (case-insensitive)

      // Use a customized Comparator for Integers
      Comparator<Integer> intComp = (i1, i2) -> i1%10 - i2%10;

      // Sort and search a "List" of Integers
      List<Integer> lst = new ArrayList<Integer>();
      lst.add(42);  // int auto-box Integer
      lst.add(21);
      lst.add(34);
      lst.add(13);
      Collections.sort(lst, intComp);
      System.out.println(lst);  //[21, 42, 13, 34]
      System.out.println(Collections.binarySearch(lst, 22, intComp)); //1
      System.out.println(Collections.binarySearch(lst, 35, intComp)); //-5 (insertion at index 4)
   }
}
```

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

HashSet, LinkedHashSet, TreeSet

```
// Interface java.util.Set<E>
abstract boolean add(E o)           // adds the specified element if it is not already present
abstract boolean remove(Object o)   // removes the specified element if it is present
abstract boolean contains(Object o) // returns true if it contains o

// Set operations
abstract boolean addAll(Collection<? extends E> c) // Set union
abstract boolean retainAll(Collection<?> c)        // Set intersection
```

- The implementations of Set<E> interface include:
  - HashSet<E>: Stores the elements in a hash table (hashed via the hashcode()). HashSet is the best all-round implementation for Set.
  - LinkedHashSet<E>: Stores the elements in a linked-list hash table for better efficiency in insertion and deletion. The element are hashed via the hashCode() and arranged in the linked list according to the insertion-order.
  - TreeSet<E>: Also implements sub-interfaces NavigableSet and SortedSet. Stores the elements in a red-black tree data structure, which are sorted and navigable. Efficient in search, add and remove operations (in O(log(n))).
- elements are check for duplication via the overridden equal().

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

PriorityQueue

```
// Interface java.util.Queue<E>
// Insertion at the end of the queue
abstract boolean add(E e)   // throws IllegalStateException if no space is currently available
abstract boolean offer(E e) // returns true if the element was added to this queue, else false

// Extract element at the head of the queue
abstract E remove()         // throws NoSuchElementException if this queue is empty
abstract E poll()           // returns the head of this queue, or null if this queue is empty

// Inspection (retrieve the element at the head, but does not remove)
abstract E element()        // throws NoSuchElementException if this queue is empty
abstract E peek()           // returns the head of this queue, or null if this queue is empty
```

```
// Interface java.util.Deque<E>
// Insertion
abstract void addFirst(E e)
abstract void addLast(E e)
abstract boolean offerFirst(E e)
abstract boolean offerLast(E e)

// Retrieve and Remove
abstract E removeFirst()
abstract E removeLast()
abstract E pollFirst()
abstract E pollLast()

// Retrieve but does not remove
abstract E getFirst()
abstract E getLast()
abstract E peekFirst()
abstract E peekLast()
```

- The Queue<E> and Deque<E> implementations include:
  - PriorityQueue<E>: A queue where the elements are ordered based on an ordering you specify, instead of FIFO.
  - ArrayDeque<E>: A queue and deque implemented as a dynamic array, similar to ArrayList<E>.
  - LinkedList<E>: The LinkedList<E> also implements the Queue<E> and Deque<E> interfaces, in additional to List<E> interface, providing a queue or deque that is implemented as a double- linked list data structure.

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

PriorityQueue

```
Comparator<Task> idComparator = Comparator.comparing(Task::id);
```

```
PriorityQueue<Task> priorityQueue = new PriorityQueue<>(idComparator);

priorityQueue.add(new Task(10001, "Task 1", 5));
priorityQueue.add(new Task(10003, "Task 3", 10));
priorityQueue.add(new Task(10002, "Task 2", 1));

while (!priorityQueue.isEmpty()) {
  System.out.println(priorityQueue.poll());
}
```

https://howtodoinjava.com/java/collections/java-priorityqueue/

HashMap<K,V>

```
// Interface java.util.Map<K,V>
abstract int size()              // Returns the number of key-value pairs
abstract boolean isEmpty()       // Returns true if this map contain no key-value pair
abstract V get(Object key)       // Returns the value of the specified key
abstract V put(K key, V value)   // Associates the specified value with the specified key
abstract boolean containsKey(Object key)     // Returns true if this map has specified key
abstract boolean containsValue(Object value) // Returns true if this map has specified value
abstract void clear()              // Removes all key-value pairs
abstract void V remove(object key) // Removes the specified key
```

The map<K,V> provides these method to allow a map to be viewed as a Collection:

```
// java.util.Map(K,V)
abstract Set<K> keySet()          // Returns a set view of the keys
abstract Collection<V> values()   // Returns a collection view of the values
abstract Set<Map.Entry<K,V>> entrySet()  // Returns a set view of the key-value
```

The nested class Map.Entry<K,V> contains these methods:

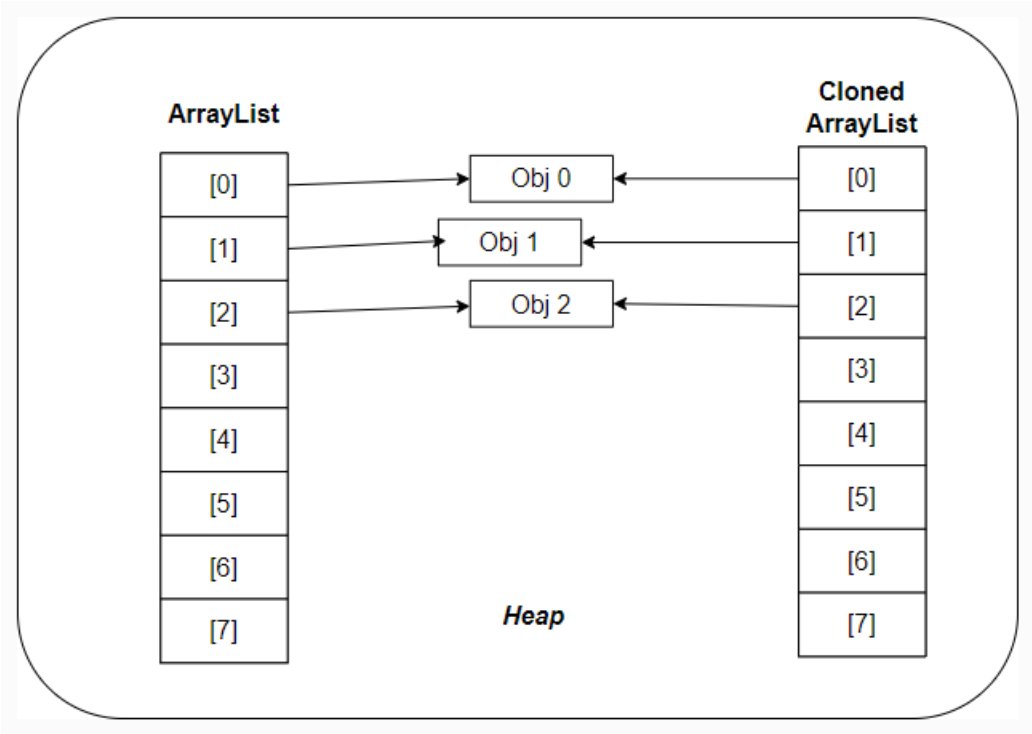
```
// Nested Class Map.Entry<K,V>
K getKey()     // Returns the key of this map entry
V getValue()   // Returns the value of this map entry
V setValue()   // Replaces the value of this map entry
```

- The implementations of Map<K,V> interface include:
  - HashMap<K,V>: Hash table implementation of the Map<K,V> interface. The best all-around implementation. Methods in HashMap is not synchronized.
  - TreeMap<K,V>: Red-black tree implementation of the SortedMap<K,V> interface.
  - LinkedHashMap<K,V>: Hash table with link-list to facilitate insertion and deletion.
  - Hashtable<K,V>: Retrofitted legacy (JDK 1.0) implementations. A synchronized hash table implementation of the Map<K,V> interface that does not allow null key or values, with legacy methods.

https://www3.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html

Shallow-Copy matter

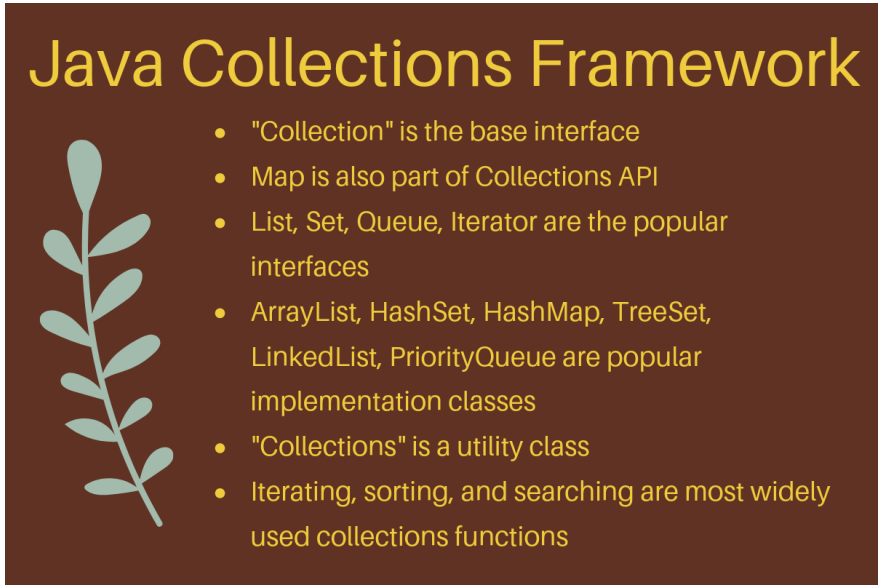
- When you create a collection type reference from an existing reference, a shallow copy is performed.
  - Copy only references



https://howtodoinjava.com/java/collections/arraylist/arraylist-clone-deep-copy/

Key Points

- 4 Categories of java Collections
  - Linear (ArrayList), Set, Map, Priority Queue (Tree-based)
  - Java collections implements **Collection** and **Iterator** interface
  - Because of Collection Interface, java collections can **be transformed from one form to another**.
  - Hash-based, set, map, has **no order** (unless new implementation provided)



https://www.digitalocean.com/community/tutorials/collections-in-java-tutorial

- Appropriate choice of data collection for your solutions tightly relates to your program performance.
  - Type of Data Structures Design effects its performance
    - E.g. linear – non-linear, contiguous – non-contiguous, etc

Collection classes in a Nutshell

Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safety
ArrayList	✅	✅	❌	✅	✅	❌
LinkedList	✅	❌	❌	✅	✅	❌
HashSet	❌	❌	❌	❌	✅	❌
TreeSet	✅	❌	❌	❌	❌	❌
HashMap	❌	✅	✅	❌	✅	❌
TreeMap	✅	✅	✅	❌	❌	❌
Vector	✅	✅	❌	✅	✅	✅
Hashtable	❌	✅	✅	❌	❌	✅
Stack	✅	❌	❌	✅	✅	✅
CopyOnWriteArrayList	✅	✅	❌	✅	✅	✅
ConcurrentHashMap	❌	✅	✅	❌	❌	✅
CopyOnWriteArraySet	❌	❌	❌	❌	✅	✅

- 👉 ArrayList
- 👉 HashSet
- 👉 HashMap

https://www.digitalocean.com/community/tutorials/collections-in-java-tutorial