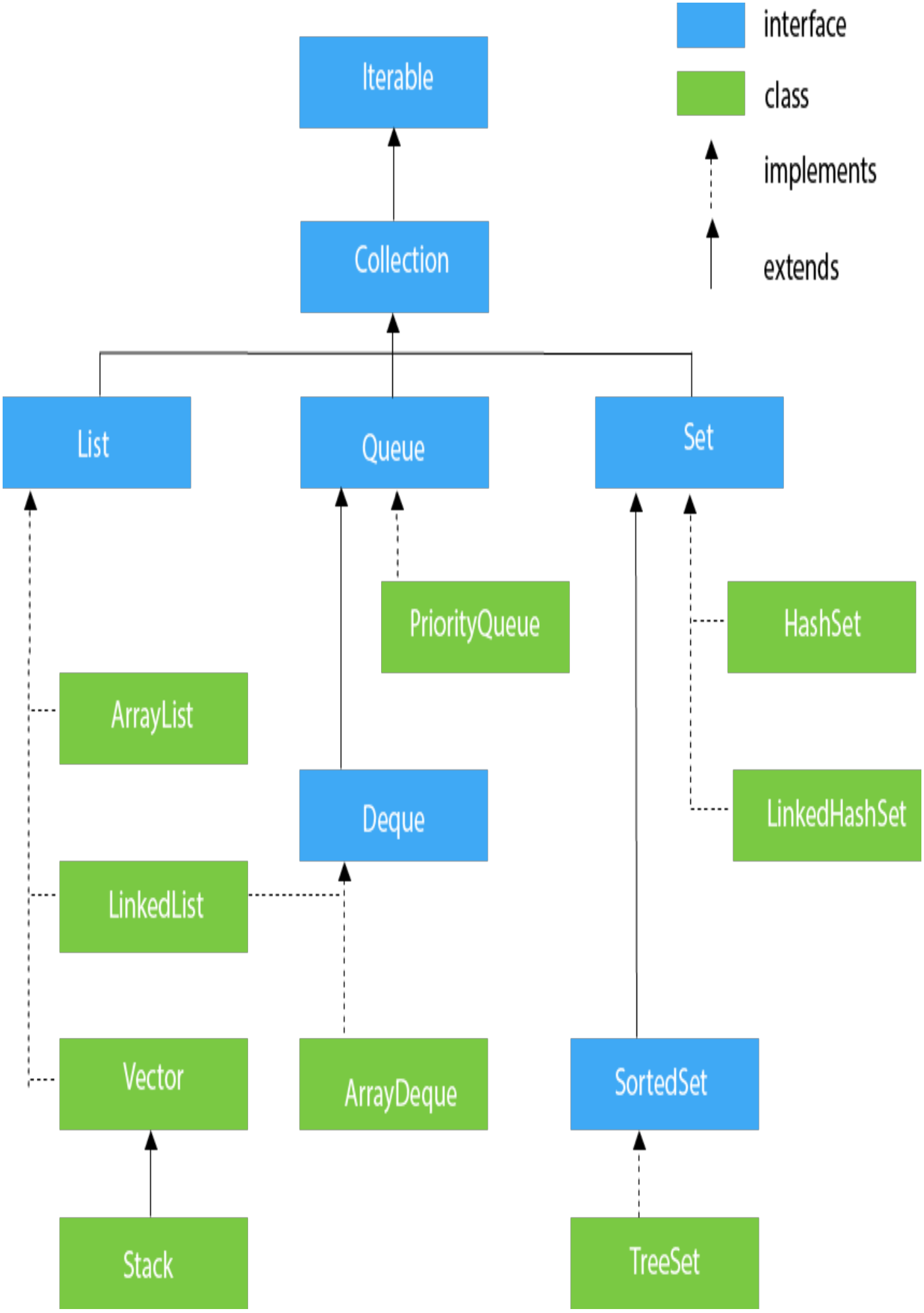
interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Collection

- Collection is a group of references (Objects), represented as one entity.
- Advantage of the collection framework is it supports Heterogenous data.
- Every collection has one underlying data structure.
- Collection has inbuilt methods, very useful in performing the basic to advance operations on the data.
- Collection reduces the effort of building the data structure in a traditional manner.

Collection Framework

- It is a group of classes and interfaces present in the **java.util** package
- Used to perform operations on dynamic data.

List

1. It is an interface which extends Collection interface.
2. List cannot be instantiated.
3. Multiple null values, duplicates are allowed.
4. Insertion order is preserved.
5. Indexed type collection.

Array List

- Implements **List, Collection, Iterable, Seralizable, Cloneable** and **RandomAccess** interfaces.

Methods.

1. add(data→Object Element) → Appends(will insert) the element to the end of an array list.
2. size() → Returns the number of elements in the list. Return type is **int.**
3. get(index.no) → Returns the element or data in the list at the specified index.
4. set(indexno, data→Object Element) → **Replaces** the element at the specified index.
5. isEmpty() → Returns **Boolean** type . Checks whether the array list is empty or not.
6. remove(int index) → removes the element at the specified index.

****

Differences….

1. List<Object> list= new ArrayList<>(); //Storing class Object in Interface reference
2. ArrayList<Object> list1=new ArrayList<>(); //Storing class Object in class reference

1st will **Emphasize** on the **Abstraction.** Whereas 2nd is not.

1st is **flexible**, then 2nd. → Actual implementation of list can be changed without modifying the rest of the code.

But if we want some methods dedicated to that particular class then 2nd is preferred

LinkedList

Implements **List, Dequeue, Collection, Iterable, Seralizable, Cloneable** and **RandomAccess** interfaces.

Methods.

1. addFirst() → Adds the data to the start of the LinkedList.
2. addLast() → Adds the data to the end of the LinkedList.
3. add(Index_no,data) → Adds the data to the specified index.
4. getFirst()→ Returns the first element of a LinkedList.
5. getLast()→ Returns the last element of a LinkedList.
6. get(Index) → Returns the element at the specified index
7. removeFirst() → Removes the first element from the List
8. removeLast() → Removes the last element from the List
9. size() → returns the Size of a list

Vector

Vector implements **List**, **Collection**, **Iterable**, **Seralizable**, **Cloneable** and **RandomAccess** interfaces

It's the only class in Vector which is Thread safe i.e Synchronized

Vector contains legacy methods which are not a part of a collection frame work eg. include Capacity(), firstElement(), removeElement() etc.

Methods

1. add(): Adds an element to the end of the vector
2. addAll(): Adds all the elements of a collection to the end of the vector
3. capacity(): Returns the current capacity of the vector
4. clear(): Removes all elements from the vector
5. contains(): Returns true if the vector contains a specified element
6. containsAll(): Returns true if the vector contains all the elements of a specified collection
7. copyInto(): Copies the components of the vector into a specified array
8. equals(): Compares the vector with a specified object for equality
9. ensureCapacity(): Increases the capacity of the vector if necessary
10. firstElement(): Returns the first element of the vector
11. get(): Returns the element at a specified position in the vector
12. lastElement(): Returns the last element of the vector
13. remove(index): Removes an element from a specified position
14. removeAll(): Removes all the elements from the vector
15. set(): Modifies an element at a specified position in the vector
16. trimToSize(): Removes excess capacity and keeps the capacity equal to the size

Difference between Vector and ArrayList

Vectors are synchronized and Thread safe whereas ArrayList are not
If Vectors reaches the capacity it doubles the size to 100% whereas ArrayList only 50%

Stacks

It extends the Vector class and implements **List**, **Collection**, **Iterable**, **Seralizable**, **Cloneable** and **RandomAccess** interface.

Note: Since stack is extending the Vector class all the methods of the Vector class can be accessed using the Stack object (Super class, Sub class i.e Inheritance concept).

Methods

1. empty() → returns Boolean → True if stack is empty; False if stack is not empty
2. push() → Inserts the element to the top of a stack
3. pop() → Removes the element from the top of the stack
4. peek() → Points to the element at the top of the stack
5. Search() → Returns int → Index position of the specified index
6. Size() → Returns int → Returns the size of a stack

Queues

It extends the **Collection**, **Iterable**, **Seralizable**, **Cloneable** and **RandomAccess** interface.

Methods

1. add() → adds elements to the queue
2. element() →Returns the head of the queue throws NoSuchElement Exception if queue is empty
3. peek() → Returns the head of the queue, returns null if queue is empty
4. remove() → Removes the head of the queue throws NoSuchElement Exception if queue is empty
5. poll() → Removes the head of a queue returns null if queue is empty

ProrityQueue

It implements **Queues, Collection**, **Iterable**, **Seralizable**, **Cloneable** and **RandomAccess** interface.

Its internal implementation is **Min-Heaps**

Note: Will discuss the difference of iterating the priority queue using loops vs iterator interface….
The difference is as follows

- PriorityQueue uses a **heap structure** internally
- iterator() just gives you the elements in the **heap array**, not in priority order

Set

1. Set do not allow duplicates.
2. Set do not have index.
3. Only one null value can be stored.
4. Insertion order is not preserved.

Operations performed on the Set.

Insertion→ add(data) →adds data to the set.

Search→ contain(data) → Checks if the data is contained in the set or not. → Returns Boolean result.

Delete → remove(data) → Removes the specified data.

10 20 30 40 →Set

PRINT A SET

10 20 30 40 → Insertion order is not preserved

20 40 10 30

****

Iterator → it is an Object used to traverse the set elements.

It has two methods

- next() → Returns next element in a collection.
- hasNext() → Returns Boolean result. **True:** if there are more elements to iterate. **False:** if there are no more elements to iterate.

HashSet

Internal working of the HashSet is based on HashMap.

Implements Set,Cloneable, Serializable, interfaces and extends AbstractSet class.

Methods

add(data) →Returns **Boolean** type. Adds the specified element to a set.

remove(Object o) → Returns **Boolean** type. Removes the element from the set.

contains(Object o) → Return **Boolean** type. Check whether the element is present in the set or not.

clear() → Removes all the elements from the set.

isEmpty() → Returns **Boolean** type . Checks whether the set is empty or not.

clone() → Creates a shallow copy of the set.

size( ) → Returns the size of the set.

TreeSet

Internal working of TreeSet is Red black tree(Self balancing Binary search tree)

HashMap

- HashMap is a class which implements the Map interface.
- It works on a mechanism of the key and value.
- Key cannot be duplicate if duplicate Overwrites the existing values.
- Values can be duplicate.
- Null can be used as key and values

Operations

- Insert: put(key, value) → Used to insert the element in a Map.
- Retrieve: get(key) →Used to Retrieve the key value of a specified key. Returns null if key does not exist.
- Remove: remove(key) → used to remove key and the value corresponding to the key in a map.
- Iteration:
    - Syntax using for each loop:
        - for(Map.Entry<Key, Value> Iterator/value: Object_ref.entrySet())

contains(Key) → returns Boolean value. **True:** if key is present. **False:** if key is not present.

keySet()

index = (n - 1) & hash