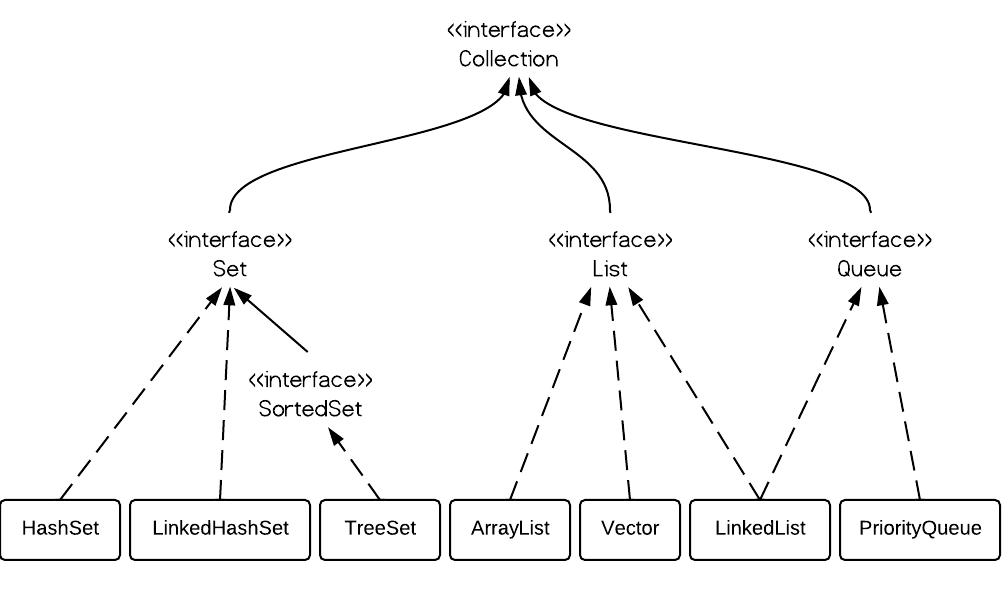
JDK API Question

1. LinkedList vs ArrayList



\* ArrayList: Basically array, dynamic increasing 50%, not thread safe. Get/Set fast because it is array.

\* LinkedList: Doubly linkedlist implemented, so it has poll(), peek(), offer(); Remove/Add fast. Get/Set slow.

\* Vector: array, dynamic increasing 100% size, thread safe.

Vector vs ArrayList: Vector is slow and take more space. Most of time, developer use locks to lock a few instruction, so does not need thread-safe data structure like vector.

2.Create two dimensional List, which syntax is correct?

The following are ok.

List<List<Integer>> table4 = new ArrayList<List<Integer>>();

List<List<Integer>> table1 = new ArrayList<List<Integer>>();

List<ArrayList<Integer>> table2 = new ArrayList<ArrayList<Integer>>();

ArrayList<ArrayList<Integer>> table3 = new ArrayList<ArrayList<Integer>>();

The following is wrong,

List<**List**<Integer>> table = new ArrayList<ArrayList<Integer>>();

Rule of thumb, the inner type must be identical from left to right. The outter type in the rhs needs to be solid type not interface.

3. Iterate map

There are a few ways to do this,

\* EntrySet and loop,

public void iterateUsingEntrySet(Map<String, Integer> map) {

    for (Map.Entry<String, Integer> entry : map.entrySet()) {

        System.out.println(entry.getKey() + ":" + entry.getValue());

    }

}

\* Using keySet()

for (Integer key : customers.keySet()) {

System.out.println("Key : " + key + " value : " + customers.get(key));

}

\* EntrySet and Iterator

public void iterateUsingIteratorAndEntry(Map<String, Integer> map) {

    Iterator<Map.Entry<String, Integer>> iterator = map.entrySet().iterator();

    while (iterator.hasNext()) {

        Map.Entry<String, Integer> entry = iterator.next();

        System.out.println(entry.getKey() + ":" + entry.getValue());

    }

}

\* KeySet and Iterator

Iterator<Integer> iterator = customers.keySet().iterator();

while (iterator.hasNext()) {

Integer key = iterator.next();

System.out.println("Key : " + key + " value : " + customers.get(key));

}

\* With Lambda(forEach)

public void iterateUsingLambda(Map<String, Integer> map) {

    map.forEach((k, v) -> System.out.println((k + ":" + v)));

}

\* Stream API

public void iterateUsingStreamAPI(Map<String, Integer> map) {

    map.entrySet().stream()

      // ...

      .forEach(e -> System.out.println(e.getKey() + ":" + e.getValue()));

}

Another example,

map.entrySet().stream()

.filter(x -> "Jan".equals(x.getValue()))

.forEach( x -> System.out.println("Key : " + x.getKey() + " Value : " + x.getValue()));

1/ Sort Map by Key

Use treemap, first build a hashmap, and then add hashmap object as parameter to treemap constructor.

Map<String, String> unsortMap = new HashMap<String, String>();

Map<String, String> treeMap = new TreeMap<String, String>(unsortMap);

Then when you iterator each entry, their key is sorted.

2/ Sort Map by Value

Still uses TreeMap, but you have to implement a comparator object.

Map<Integer, String> treeMap = new TreeMap<Integer, String>(

new Comparator<Integer>() {

@Override

public int compare(Integer o1, Integer o2) {

return o2.compareTo(o1);

}

});

/\* For Java 8, try this lambda

Map<Integer, String> treeMap = new TreeMap<>(

(Comparator<Integer>) (o1, o2) -> o2.compareTo(o1)

);

\*/

treeMap.putAll(unsortMap);

printMap(treeMap);

4. Collections.sort() usage.

Sort a list of object by its property value.

**public** **static** **void** main(String[] args) **throws** Exception {

List<Movie> m = **new** ArrayList<>();

m.add(**new** Movie("a", 3));

m.add(**new** Movie("b", 5));

m.add(**new** Movie("c", 4));

m.add(**new** Movie("d", 1));

Collections.*sort*(m, **new** Comparator<Movie>() {

**public** **int** compare(Movie m1, Movie m2) {

**return** (**int**) (m1.rate - m2.rate);

}

});

m.forEach(e-> System.***out***.println(e.rate));

}

5.　java.util.Arrays.sort(...)

public static void sort(int[] arr, int from\_Index, int to\_Index)

eg:

**int**[] arr = {13, 7, 6, 45, 21, 9, 101, 102};

Arrays.sort(arr);

6. java.util.Arrays.asList

public static <T> List<T> asList(T... a)

eg:

String a[] = new String[]{"abc","klm","xyz","pqr"};

List list1 = Arrays.asList(a);

7. ArrayDeque vs Linkedlist

8. The usage of priority queue.

9. HashMap, LinkedHashMap and HashMap’s implementation

10. Autoclosable?