Java Collection Framework

A Java collection framework provides an architecture to store and manipulate a group of objects.

Components of Java Collection Framework:

A Java collection framework includes the following:

- ❖ Interfaces
- ❖ Classes
- ❖ Algorithm

Java Collection Framework

Interfaces:
Interface in Java refers to the abstract data types. They allow Java collections to be manipulated independently from the details of their representation. Also, they form a hierarchy in object-oriented programming languages.

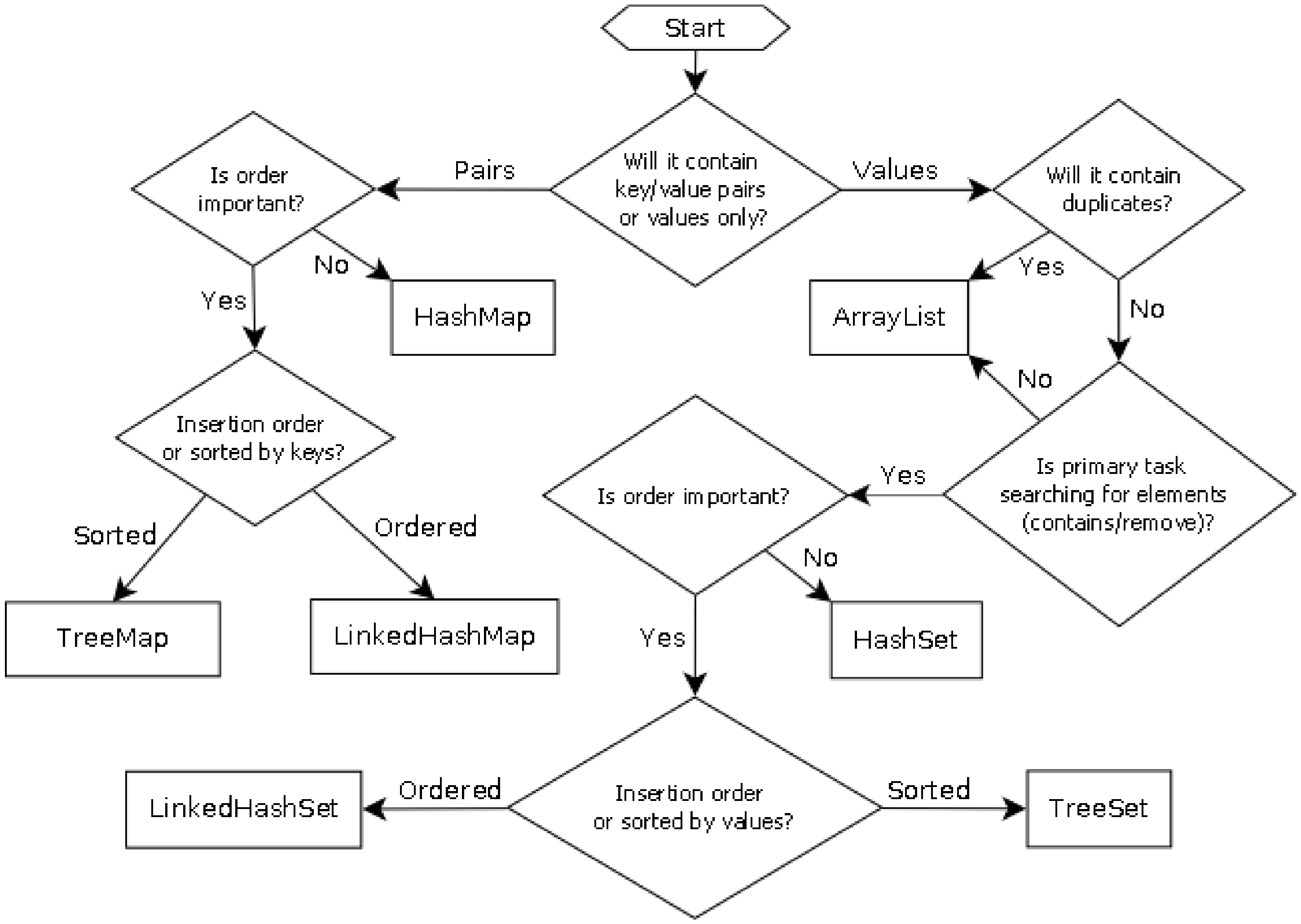
Classes:
Classes in Java are the implementation of the collection interface. It basically refers to the data structures that are used again and again.

Java Collection Framework

Algorithm:

Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces. Algorithms are polymorphic in nature as the same method can be used to take many forms or you can say perform different implementations of the Java collection interface.

Choosing a collection



Start
Will it contain key/value pairs or values only?
Pairs
Values
Is order important?
No
HashMap
Yes
Insertion order or sorted by keys?
Sorted
TreeMap
Ordered
LinkedHashMap
Will it contain duplicates?
Yes
ArrayList
No
Is primary task searching for elements (contains/remove)?
No
Yes
Is order important?
No
HashSet
Yes
Insertion order or sorted by values?
Ordered
LinkedHashSet
Sorted
TreeSet

- see also: http://initbinder.com/bunker/wp-content/uploads/2011/03/collections.png

Difference between List, Set, and Map in Java

List	Set	Map
The list interface allows duplicate elements	Set does not allow duplicate elements.	The map does not allow duplicate elements
The list maintains insertion order.	Set do not maintain any insertion order.	The map also does not maintain any insertion order.
We can add any number of null values.	But in set almost only one null value.	The map allows a single null key at most and any number of null values.
List implementation classes are Array List, LinkedList.	Set implementation classes are HashSet, LinkedHashSet, and TreeSet.	Map implementation classes are HashMap, HashTable, Tree Map, ConcurrentHashMap, and LinkedHashMap.
If you need to access the elements frequently by using the index then we can use the list	If you want to create a collection of unique elements then we can use set	If you want to store the data in the form of key/value pair then we can use the map.

List Example

```
// Creating a List
List<String> al = new ArrayList<>();

// Adding elements in the List
al.add("mango");
al.add("orange");
al.add("Grapes");

// Iterating the List
// element using for-each loop
for (String fruit : al)
    System.out.println(fruit);
```

Set Example

```java
// Set demonstration using HashSet
Set<String> Set = new HashSet<String>();

// Adding Elements
Set.add("one");
Set.add("two");
Set.add("three");
Set.add("four");
Set.add("five");

// Set follows unordered way.
System.out.println(Set);
```

Map Example

```
// Creating object for Map.
Map<Integer, String> map
    = new HashMap<Integer, String>();

// Adding Elements using Map.
map.put(100, "Amit");
map.put(101, "Vijay");
map.put(102, "Rahul");

// Elements can traverse in any order
for (Map.Entry m : map.entrySet()) {
    System.out.println(m.getKey() + " "
                       + m.getValue());
}
```

Collections summary

collection	ordering	benefits	weaknesses
array	by index	fast; simple	little functionality; cannot resize
`ArrayList`	by insertion, by index	random access; fast to modify at end	slow to modify in middle/front
`LinkedList`	by insertion, by index	fast to modify at both ends	poor random access
`TreeSet`	sorted order	sorted; O(log N)	must be comparable
`HashSet`	unpredictable	very fast; O(1)	unordered
`LinkedHashSet`	order of insertion	very fast; O(1)	uses extra memory
`TreeMap`	sorted order	sorted; O(log N)	must be comparable
`HashMap`	unpredictable	very fast; O(1)	unordered
`LinkedHashMap`	order of insertion	very fast; O(1)	uses extra memory
`PriorityQueue`	natural/comparable	fast ordered access	must be comparable

- It is important to be able to choose a collection properly based on the capabilities needed and constraints of the problem to solve.