JCF notes

Prof Bill, Feb 2020

Reading:

- Official Java stuff, docs.oracle.com/javase/tutorial/collections
- Collection interface in Java API (note the methods!), docs.oracle.com/javase/8/docs/api/java/util/Collection.html
- This is a little more palatable source; I like (and borrowed) his figures, dzone.com/articles/an-introduction-to-the-java-collections-framework

Thank you - I (again) bow down to Noctrl's own **Dr. Godfrey Muganda** for his excellent Java textbook. I have liberally borrowed ideas from this book.

media.pearsoncmg.com/bc/abp/cs-resources/products/product.html#product,isbn=0134038177

Sections: 1) Intro, 2) Lists, 3) Sets, 4) Maps, 5) Collections methods, 6) Code snippets

Terms:

JCF Util package collection list set map	generic enhanced for loop foreach lambda function key-value pairs
ArrayList LinkedList HashSet TreeSet HashMap TreeMap	hashCode() compareTo() Comparable Comparator

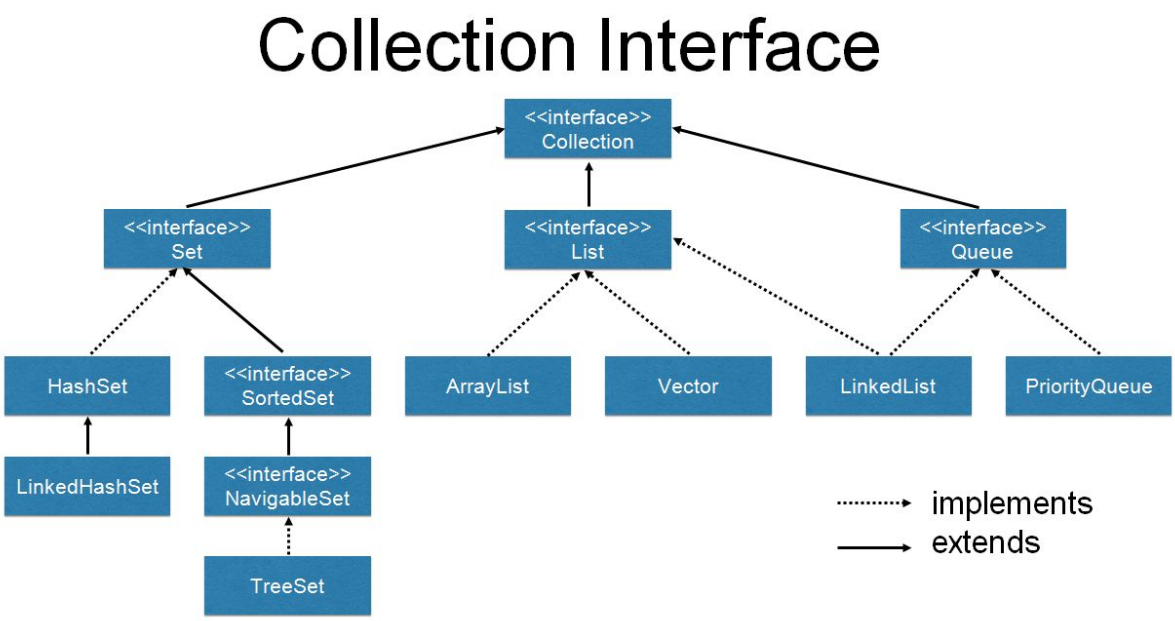
1. Intro to the JCF

Java Collections Framework = **JCF**
collection: object that contain other objects
3 types of collections: list, set, map

- **list** - ordered collection
- **set** - unordered, no duplicates
- **map** - key-value pairs, quick retrieval by key

JCF is **generic**, so Collection<T>
JCF is part of the **util package** in Java. This Javadoc is invaluable when coding!
docs.oracle.com/javase/10/docs/api/index.html?java/util/package-summary.html

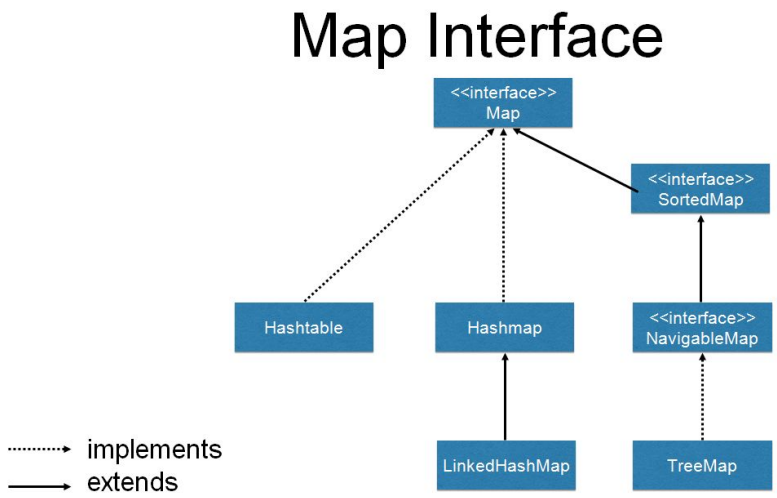
UML for class hierarchy - part 1: Collection interface, List and Set
Source: dzone.com/articles/an-introduction-to-the-java-collections-framework



Collection interface methods:

> add(Object), addAll(Collection), clear(),
> contains(Object), containsAll(Collection), isEmpty(),
> remove(Object), removeAll(Collection), size(), stream(), toArray()

Part 2 is Map, which has a radically different set methods; little overlap with Collection, so Map has its own interface:



Iterator for detailed control over looping

Easier and more common, use the **enhanced for loop** (built using Java Iterator)

```
for( String name: nameList) {
    // do something here to each name in the list
}
```

Another way, Java **foreach** loop with a **lambda function**:

```
nameList.forEach(
    x ->
        {
            System.out.println("%s %d\n", x, x.length);
        });
```

2. Lists

Two Lists: ArrayList, LinkedList

List interface methods: 1) inherits all Collection methods, and 2) and adds these:

> add(int pos), addAll(Collection), get(i), indexOf(Object),
> remove(int pos), sort(Comparator)

ArrayList, LinkedList is-a List; can use the super class in declaration (polymorphism)

```
List<String> nameList = new ArrayList<>();
```

ListIterator methods give detailed control over iteration of List

---

3. Sets

sets are **unordered** collections with **no duplicates**

Java says:

> A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
>
> - docs.oracle.com/javase/tutorial/collections/interfaces/set.html

Different implementations of Set:

- **HashSet -** is-a Set, implements Set with a hash table, uses **hashcode()** method, inherited from Object
- **LinkedHashSet -** it's HashSet with a linked list added to preserve order (meh)
- **TreeSet** - implements Set with a binary tree

SortedSet - not an implementation, an interface for sorting elements in a Set, example: TreeSet is-a SortedSet.

4. Maps

maps store **(key, value) pairs**; each key has one value; key -> value access is fast
Java says:

> A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.
> The Java platform contains three general-purpose Map implementations: HashMap, TreeMap, and LinkedHashMap. Their behavior and performance are precisely analogous to HashSet, TreeSet, and LinkedHashSet, as described in The Set Interface section.
>
> - docs.oracle.com/javase/tutorial/collections/interfaces/map.html

Catch that? Implementations are similar to Set.
They're **HashMap**, **TreeMap**, and **LinkedHashMap**. **SortedMap** interface, too.

Some new methods in Map (types are K=key, V=value):

> V get(K) - get value for this key
> put(K, V) - put (key, value) in map
> V remove(K) - remove (key, value) from map
> Set<K> keySet() - create set of all keys in map
> Collection<V> values() - create collection of all values in map
> And… containsKey(K), containsValue(V), clear(), isEmpty()

---

5. Collections methods

These are some **very useful** static methods for Collection objects!
docs.oracle.com/javase/10/docs/api/index.html?java/util/Collections.html

Most popular methods are:

- binarySearch() - with Comparable or Comparator
- sort() - with Comparable or Comparator
- max(), min() - with Comparable or Comparator

- copy()
- reverse() - reverse the order of elements
- shuffle() - randomize!

Comparable vs. Comparator

This is important - 2 ways to compare objects (for sorting, searching, everything!):

- **compareTo()** method, inherited from **Comparable** interface
- **Comparator** interface

This is a nice example to walk through… Player class, 1) order by rank using Comparable, and 2) order by name, rank, or age using Comparator.

www.baeldung.com/java-comparator-comparable

Comparable	Comparator
1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc.	Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.
2) Comparable **affects the original class** i.e. actual class is modified.	Comparator **doesn't affect the original class** i.e. actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is found in **java.lang** package.	Comparator is found in **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method.

Source: www.javatpoint.com/difference-between-comparable-and-comparator

Grudge match: Comparable vs. Comparator!

- The advantage of Comparable: easy to create and called implicitly.
- The advantage of Comparator: more flexible, more control, can have many ways to compare objects of a class (name, age, rank, serial number, etc)

6. Code snippets

ArrayList - probably the most popular JCF class;

```
ArrayList<Dog> thePound = new ArrayList<>(); // create

Dog fido = new Dog( "Fido", "Schipperke"); // add
thePound.add( fido);

Dog dog7 = thePound.get(7); // get 7th dog

for( Dog d : thePound) { // for each dog in the pound
      findOwner( d);
}

System.out.println( the_pound); // print, Dog must have toString()

thePound.sort( cmp); // must create Comparator cmp
```

TreeSet - built using a binary search tree (BST); objects must be Comparable

```
Set<String> fruity = new TreeSet<>(); // create

fruity.add( "apple"); // add
fruity.add( "banana");
fruity.add( "grapes");
fruity.add( "apple"); // add will fail! no dups allowed

fruity.contains( "orange"); // false
fruity.remove( "grapes"); // remove from set
```

HashMap - the keys in your HashMap should have hashCode() defined (String does)

```
Map<String, Car> usedCars = new HashMap<>(); // create

usedCars.put( "Mustang", car1); // put = add
usedCars.put( "Subaru", orangeCar);

usedCars.get( "Subaru"); // returns the orangeCar object
usedCars.containsKey( "Corvette"); // false
```

/* remember - use google to find more examples if you need them! */