

Chapter 22 Java Collections Framework

1. The Java Collections Framework defines the Java API for handling common data structures tasks in Java. It defines classes and interfaces for storing and manipulating data in sets, lists, and maps.

A convenience class is an abstract class that partially implements an interface. The Java Collections Framework defines interfaces, convenience abstract classes, and concrete classes.

2. Yes. The concrete classes of Set, List, and Map implements the clone() method in the Cloneable interface.
3. By defining these two methods in the Collection interface, the instances of Collection can use these two methods. For instance, you may have a method with a parameter of the Collection type. This parameter can use the hashCode method and the equals method since the methods are in the Collection interface.
4. The default implementation of the equals method is to compare the references of this object with the other. The default implementation of the hashCode method returns the object's memory address of the object.
5. The Set is an interface. To create an instance of Set, you need to use HashSet or TreeSet. To insert an element to a set, use the add method. To remove an element from a set, use the remove method. To find the size of a set, use the size() method.
6. HashSet is unsorted, but TreeSet is sorted. HashSet is more efficient than TreeSet if you don't want the elements in a set to be sorted. If you create a TreeSet using its default constructor, the compareTo method is used to compare the elements in the set, assuming that the class of the elements implements the Comparable interface. To use a comparator, you have to use the constructor TreeSet(Comparator comparator) to create a sorted set that uses the compare method in the comparator to order the elements in the set. A runtime error would occur if you add an element that cannot be compared with the existing elements in the tree set?
7. To traverse a set, use the iterator method to obtain an iterator. You can then traverse the set through the iterator. Using an iterator, you can traverse only sequentially from the beginning to the end.
8. To sort the elements in a set using the Comparable interface, there are two ways: (1) create a TreeSet using new TreeSet(), (2) create a TreeSet from a set using new TreeSet(set). To sort the elements in a set using the Comparator interface, create a TreeSet using new TreeSet(Comparator), then add the elements to the tree set.

9. set2 is not changed by all these methods. (If you use HashSet, the order of the elements in the set is unpredictable.)

What is set1 and set2 after executing set1.addAll(set2);

set1 is [green, red, blue, yellow]

What is set1 and set2 after executing set1.add(set2);

set1 is [green, red, yellow, [red, blue, yellow]]

Note set2 is added as a whole element into set1.

What is set1 and set2 after executing set1.removeAll(set2);

set1 is [green]

What is set1 and set2 after executing set1.remove(set2);

set1 is [green, red, yellow]

What is set1 and set2 after executing set1.retainAll(set2);

set1 is [red, yellow]

What is set1 after executing set1.clear();

10. The Comparable interface contains the compareTo method and Comparator interface contains the compare method and equals method. Normally, if the objects of a class have natural order (e.g., String, Date), let the class implement the Comparable interface. The Comparator interface is more flexible in the sense that it enables you to define a new class that contains the compare(Object, Object) method to compare two objects of other classes.

The Comparable interface is in the java.lang package, and the Comparator interface is in the java.util package.

11. Since The equals method is also defined in the Object class. Therefore, you will not get a compilation error if you don't implement the equals method in your custom comparator class. However, in some cases implementing this method may improve performance by allowing programs to determine that two distinct comparators impose the same order.

12. Use the add or remove method to add or remove elements from a list. Use the listIterator() to obtain an iterator. This iterator allows you to traverse the list bi-directional.

13. set2 is not changed by all these methods.

What is list1 and list2 after executing set1.addAll(list2);

list1 is [red, yellow, green, red, yellow, blue]

What is list1 and list2 after executing list1.add(list2);

list1 is [red, yellow, green, [red, yellow, blue]]

What is list1 and list2 after executing list1.removeAll(list2);

list1 is [green]

What is list1 and list2 after executing list1.remove(list2);

list1 is [red, yellow, green]

What is list1 and list2 after executing list1.retainAll(list2);

list1 is [red, yellow]

What is list1 after executing list1.clear();

list1 is empty

14. ArrayList and LinkedList can be operated similarly. The critical differences between them are their internal implementation, which impacts the performance. ArrayList is efficient for retrieving elements, and for adding and removing elements from the end of the list. LinkedList is efficient for adding and removing elements anywhere in the list.
15. A simple way to create a set or a list from an array of objects is to use `new HashSet(Arrays.asList(arrayObject))` and `new ArrayList(Arrays.asList(arrayObject))`. You may also substitute HashSet with `LinkedHashSet` or `TreeSet`, and substitute ArrayList with `LinkedList`.
16. Yes.
17. The methods for lists are: `sort`, `binarySearch`, `reverse`, `shuffle`
The methods for collections are: `max`, `min`, `disjoint`, `frequency`
18.
[blue, green, red, yellow]
[white, black, green, blue]
false
true
2
19. You can use `Collections.sort(list)` to sort an ArrayList or a LinkedList and use `Arrays.sort(Object[])` to sort an array of strings.
20. You can use `Collections.binary(list, key)` to perform binary search for an ArrayList or a LinkedList and use `Arrays.binary(Object[], key)` to sort an array of strings.
21. `Collections.max(Arrays.asList(arrayObject))`
22. Vector is the same as ArrayList except that, except that Vector contains the synchronized methods for accessing and modifying the vector. Since Vector

- implements List, you can use the methods in List to add, remove elements from a vector, and use the size() method to find the size of a vector. To create a vector, use either its constructors.
23. Stack is a subclass of Vector. The Stack class represents a last-in-first-out stack of objects. The elements are accessed only from the top of the stack. You can retrieve, insert, or remove an element from the top of the stack. To add a new element to a stack, use the push method. To remove an element from the top of the stack, use the method pop. To find a stack size, use the size() method.
 24. Yes, because all the methods in the HashSet class are defined in the Collection interface.
 25. java.util.Queue is a subinterface of java.util.Collection, and LinkedList implements Queue.
 26. Use the constructors of PriorityQueue to create priority queues. By default, the elements in a priority queue are ordered in their natural order using the compareTo method in the Comparable interface. The element with the least value is assigned the highest priority in PriorityQueue.
 27. `new PriorityQueue(initialCapacity, Collections.reverseOrder())`.
 28. Map is an interface. To create an instance of Map, you need to use the HashMap class or the TreeMap class. The HashMap and TreeMap have various constructors that you can use to create a HashMap or a TreeMap. You can use the put method to add an entry to a map, and the remove method to remove an entry with the specified key from the map. Use the size method to find the size of a map.
 29. The HashMap, LinkedHashMap, and TreeMap classes are three concrete implementations of the Map interface. The HashMap class is efficient for locating a value, inserting a mapping, and deleting a mapping. The entries in a HashMap are not ordered, but the entries in a LinkedHashMap can be retrieved in the order in which they were inserted into the map (known as the *insertion order*), or the order in which they were last accessed, from least recently accessed to most recently (*access order*). The TreeMap class, implementing SortedMap, is efficient for traversing the keys in a sorted order.
 30.
 - (1) { 123=Steve Yao, 111=George Smith, 222=Steve Yao }
 - (2) { 111=George Smith, 123=Steve Yao, 222=Steve Yao }