Java Knowledge

I. String and Arrays

String immutability

Once a *string* is created *in memory(heap)*, it cannot be changed. All methods of String do not change the string itself, but rather *return a new String*. If we need a string that can be modified, we should use StringBuffer or StringBuilder (StringBuffer is synchronized, which means it is thread-safe but slower than StringBuilder). Otherwise, there would be a lot of time wasted for Garbage Collection since each time a new String is created. String is supported by a char array. The String class contains 3 fields: char value[], int offset, int count. String is an immutable class in Java. An immutable class is simply a class whose instances cannot be modified. All information in an instance is initialized when the instance is created and the information cannot be modified.

String pool is a special storage area in *Method Area*. When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object and returning its reference.

In summary, String is designed to be *immutable for the sake of efficiency and security*.

Create String with double quotes vs. constructor

For example: String a = “abcd”; String b = “abcd”; a == b is true because a and b are referring to the same string literal in the method area. The memory references are the same. When the same string literal is created more than once, only one copy of each distinct string value is stored. This is called "*string interning*".

However, String a = new String(“abcd”); String b = new String(“abcd”); a == b is false because c and d refer to two different objects in the heap. Different objects always have different memory references.

Array length vs. String length()

An *array* is a *container object* that holds a *fixed number* of values of a single type. After an array is created, its length never changes. The array's length is available as a final instance variable length. Therefore, length can be considered as a defining attribute of an array. One array specifies the element type, the number of levels of nested arrays, and the length of the array for at least one of the levels of nesting. An array contains all the members inherited from class Object.

Check value in an array

There are four different ways to check is an array contains a value.

1. using List: Arrays.asList(arr).contains(value);
2. using Set: set = new HashSet<String>(Arrays.asList(arr)); set.contains(value);
3. using Loop: String s: arr -> s.equals(value); *Most Efficiency*
4. using Arrays.binarySearch()

Array methods

1. *Print* an array: String arr = Arrays.toString(array); System.out.println(arr)
   If print array directly, it will print the address of the array, c.f. ArrayList, which can be printed directly
2. *Concatenate* arrays: combine = ArrayUtils.addAll(arr1, arr2);

II. Common Methods

Equals() vs. hashCode()

1. If two objects are equal, then they must have the same hash code.
2. If two objects have the same hashcode, they may or may not be equal.

The idea behind a Map is to be able to find an object faster than a linear search.

Recursion vs. Interation

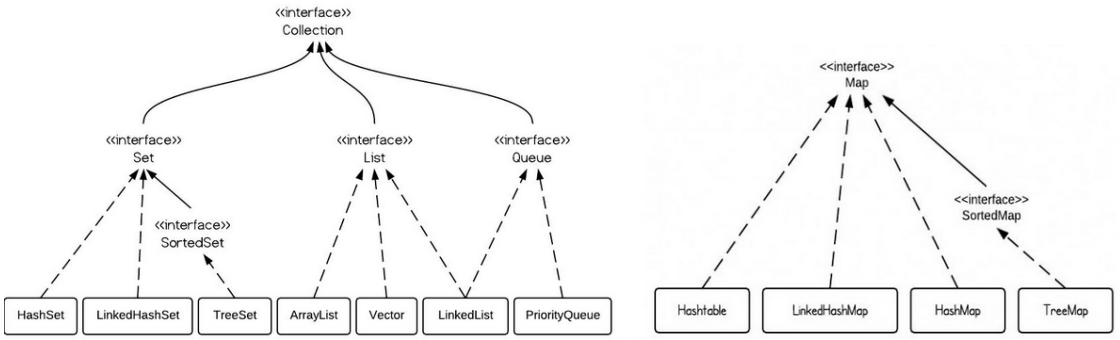
When *recursion*, the computer has to keep *track of the multiplications* to be performed later on. Which characterized by a *chain of operations*. For *iteration*, the computer only need to keep *track of the current values* of the product, whose state can be summarized by a fixed number of variables, a fixed rule that describes how the variables should be updated, and an end test that specifies conditions under which the process should terminate.

III. Collections & Generics

Collection vs. Collections

"Collection" is a root interface in the Collection hierarchy but "Collections" is a class which provide static methods to manipulate on some Collection types.

Class hierarchy of Collection:



Interfaces	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

ArrayList vs. LinkedList vs. Vector

ArrayList is implemented as a resizable array. As more elements are added to ArrayList, its size is increased dynamically. It's elements can be accessed directly by using the get and set methods, since ArrayList is essentially an array.

LinkedList is implemented as a double linked list. Its performance on add and remove is better than Arraylist, but worse on get and set methods.

Vector is similar with ArrayList, but it is synchronized.
ArrayList is a better choice if your program is thread-safe. Vector and ArrayList require more space as more elements are added. Vector each time doubles its array size, while ArrayList grow 50% of its size each time. LinkedList, however, also implements Queue interface which adds more methods than ArrayList and Vector, such as offer(), peek(), poll(), etc

	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1) amortized
remove()	O(n)	O(n)

- ArrayList has O(n) time complexity for arbitrary indices of add/remove, but O(1) for the operation at the end of the list.
- LinkedList has O(n) time complexity for arbitrary indices of add/remove, but O(1) for operations at end/beginning of the List. LinkedList is faster in add and remove, but slower in get

HashSet vs. TreeSet vs. LinkedHashSet
Set interface extends Collection interface. In a set, *no duplicates* are allowed. Every element in a set must be unique. You can simply add elements to a set, and duplicates will be removed automatically. If you need a fast set, you should use HashSet; if you need a sorted set, then TreeSet should be used; if you need a set that can be store the insertion order, LinkedHashSet should be used.
HashSet is Implemented using a hash table. Elements are *not ordered*. The add, remove, and contains methods have constant time complexity *O(1)*.
TreeSet is implemented using a tree structure(red-black tree). The elements in a set are *sorted*, but the add, remove, and contains methods has time complexity of *O(log (n))*. It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc. TreeSet is much slower because it is sorted.
LinkedHashSet is between HashSet and TreeSet. It is implemented as a hash table with a linked list running through it, so it provides the order of insertion. The time complexity of basic methods is *O(1)*.

HashMap vs. TreeMap vs. LinkedHashMap vs. Hashtable

HashMap is implemented as a hash table, and there is no ordering on keys or values.

TreeMap is implemented based on red-black tree structure, and it is ordered by the key.

LinkedHashMap preserves the insertion orde

Hashtable is synchronized, in contrast to HashMap.

Set vs. Set<?>
An unbounded *wildcard type Set<?>* can hold elements of any type, but has two boundaries.
1. Since the question mark ? stands for any type. Set<?> is capable of holding any type of elements.
2. Because we don't know the type of ?, we can't add (method) any element into Set<?>
A *raw type Set* can also hold elements of any type with no restrictions. Thus wildcard type is safe and the raw type is not.