**The advantage of Java:**

Object-oriented; Network-oriented; Large Class Library; Multithread;

High-performance; Easy-to-use;

**Create an Array, ArrayList, Stack and Queue in java.**

(1). Create a String array:

String[] myArray = new String[2];

String[] myArray = {“this”, “is”, “my”, “array”};

Create a Integer array:

int[] intArray = new int[2];

int[] intArray = {1, 2, 3, 4};

(2). Create an ArrayList:

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

myList.add(1);

(3). Create a Stack:

Stack myStack = new Stack();

(4). Create a Queue(using LinkedList):

Queue<Integer> myQueue = new LinkedList<Integer>();

myQueue.add(1);

**Primitive V.S Reference data types:**

* **primitive types** are the basic types of data
  + byte, short(2 bytes), int(4 bytes), long(8 bytes), float(4 bytes), double(8 bytes), boolean, char(2 bytes)
  + Cannot convert integer to boolean in java.
  + ASCii: A: 65; a: 97; 0: 48;
  + primitive variables store primitive values
  + 都是一些c语言中常见的数据类型，小写
  + Notes: String is not primitive type, but reference type, starting with a capital letter.
* **reference types** are any instantiable class as well as arrays
  + String, Scanner, Random, Die, int[], String[], etc.
  + reference variables store addresses
* Notes: 数据类型的转换：
  + String to Integer: Integer.parseInt(String s);
  + notes: Double, Float and Long are similar. Double.parseDouble(String s); Float.parseFloat(String s); Long.parseLong(String s)
  + int to String: String s = String.valueOf(int i); Integer.toString(int i); String s = “” + i;
  + notes: Double, Float and Long are similar.

**Primitive data type and wrapper class:**

Each primitive data type has their own primitive wrapper class.

int[] is an array of primitive int values. Integer[] is an "object" array, holding references to Integer objects.

Q: When to use primitive type and wrapper class??

A: With autoboxing, you rarely need to have both an int and an Integer variables. For most cases, use “int” is enough.

The main case where the Integer would be useful is to distinguish the case where the variable is not known (ie null). It is possible to let an object to be null, but is illegal for a primitive data type. This will also set the scene for a NullPointerException when something is being used incorrectly, which is much more programmer-friendly than some arbitrary bug down the line.

**Autoboxing V.S Unboxing**

*Autoboxing* is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called *unboxing*.

**When to use StringBuilder??**

IF you use String concatenation in a loop, you should use StringBuilder (not StringBuffer) instead of “String”, since it consumes less memory.

IF you have a single statement, you can use String cause the complier will use StringBuilder automatically.

**StringBuilder V.S StringBuffer**

StringBuilder is faster than StringBuffer because it's not synchronized.

**Create Object with or without <>(generics)**

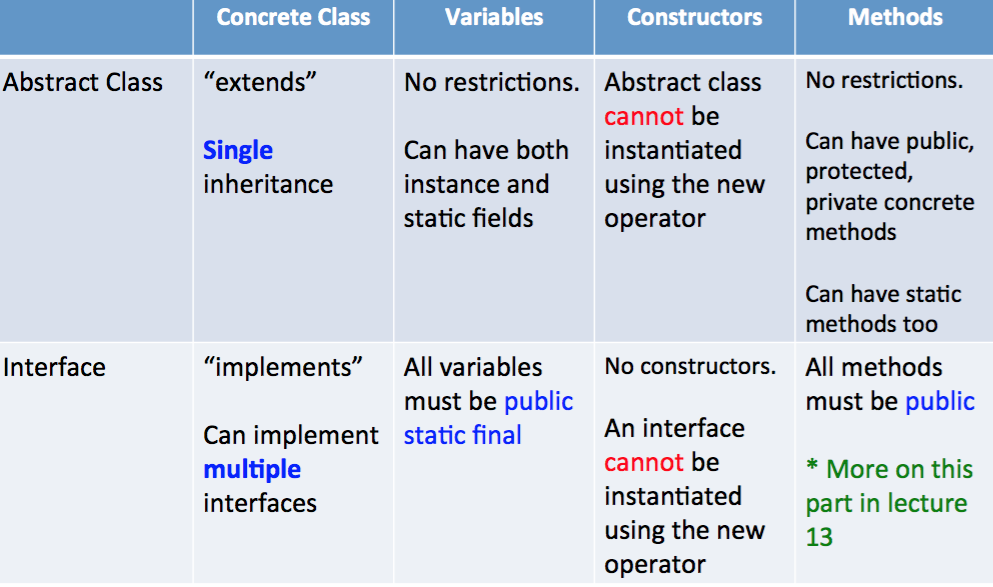
List<String> list = new LinkedList<>();

V.SList<String> list = new LinkedList();

**Generics** exist to provide compile-time protection against doing the wrong thing. The diamond operator, however, allows the right hand side of the assignment to be defined as a true generic instance with the same type parameters as the left side... without having to type those parameters again. 所以要加<>

**Abstract V.S Interface**

1. Interface should be implemented by classes, and one class could implement several interfaces.(实现多继承)
2. Abstract class could only be extended(inherited). It belongs to the the part of the java class hierarchy. (e.g. LinkedList)
3. All methods in the interface are abstract method which don’t have any implementations, while there could be some implementations in the abstract class.



**Use references to the Interface(List instead of LinkedList or ArrayList)**

e.g. **List**<String> list = new **ArrayList**<String>();

**Summary:**An interface reference variable can reference any object that implements that interface, regardless of its class type.

**2 kinds of sorting algorithm in Arrays.sort()**

    java.util.Arrays.sort(Object[]) use 2 kinds of sorting algorithms. When the length of the array less than 7, it uses **insertion** sort and when the length is greater than or equivalent to the length of the array, it uses **mergeSort**. Because An algorithm that is O(N log N) is not always faster in practice than an O(N^2) algorithm. It depends on the constants.

**Big-O**, **Big-Omega** **and** **Big-Theta**

**They are related to the growth of functions.**

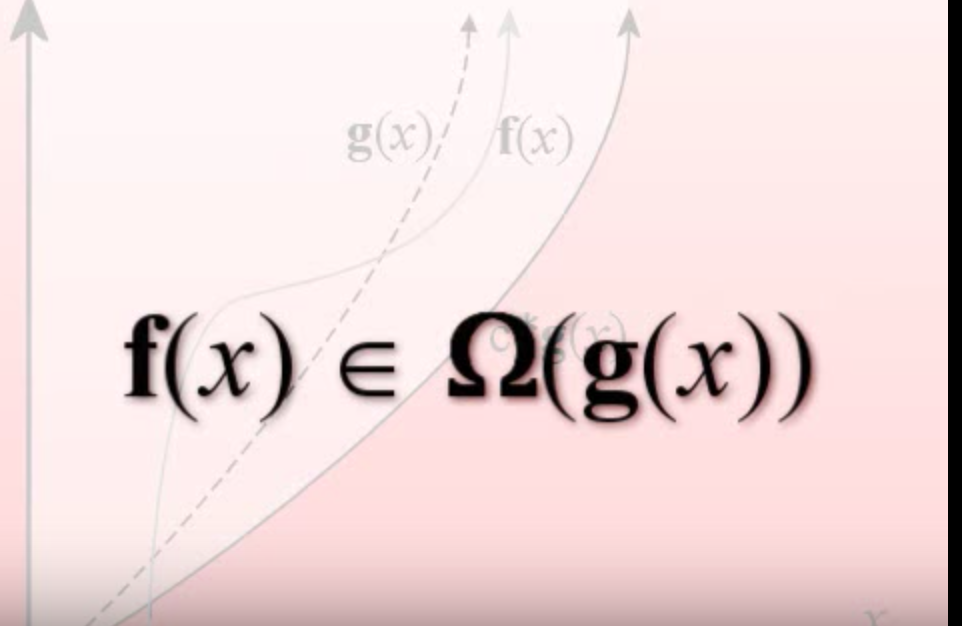
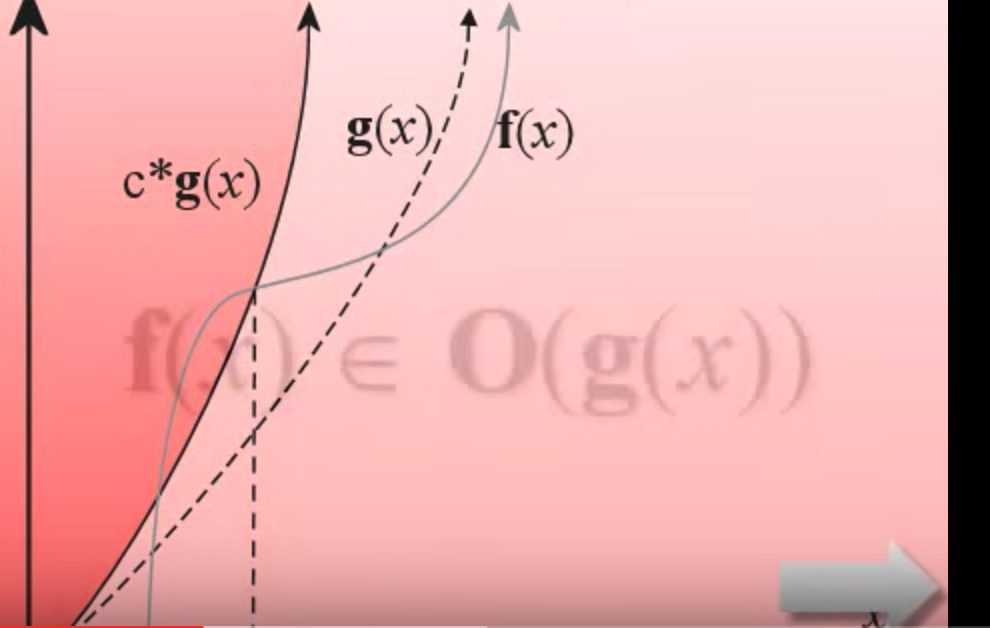
**Big-O** is a measure of the longest amount of time it could possibly take for the algorithm to complete.

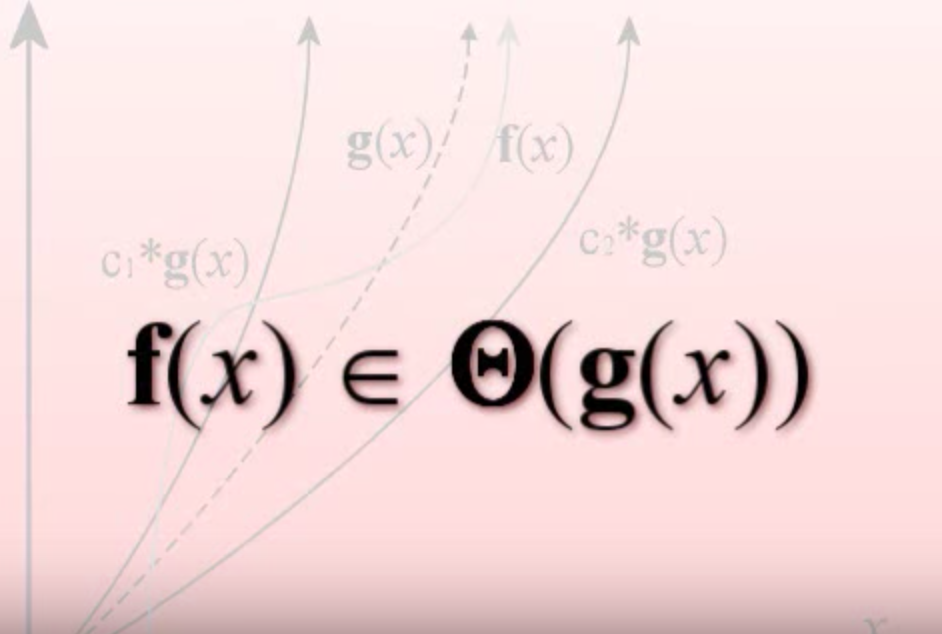
*f(n)* ≤ *cg(n), where f(n)* and *g(n) are*non-negative functions, g(n) is upper bound, then *f(n)* is Big O of *g(n)*. This is denoted as "*f(n)* = *O(g(n))*"

**Big-Omega** describes the best that can happen for a given data size.

"*f(n)* ≥ *cg(n)*", this makes *g(n)* a lower bound function

**Big-Theta** is basically saying that the function, *f(n)* is bounded both from the top and bottom by the same function, *g(n)*.





**HashMap V.S HashTable**

**1. Synchronization or Thread Safe :**  HashMap is non synchronized and not thread safe.On the other hand, HashTable is thread safe and synchronized. In other words hashmap is better for non-threading applications. HashTable should be used in multithreading applications.

**2. Null keys and null values :**  Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.

**3. Iterating the values:**  Hashmap object values are iterated by using iterator .HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.

**4.  Fail-fast iterator**  : The iterator in Hashmap is fail-fast iterator while the enumerator for Hashtable is not.

**5. Performance :**  Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized.

**ArrayList V.S Vector: Collection Question**

**1.  Synchronization and Thread-Safe**

**Vector is  synchronized while ArrayList is not synchronized.** Synchronization and thread safe means at a time only one thread can access the code .

**2.  Performance**

ArrayList is fast as it is non synchronized.

**3. Automatic Increase in Capacity**

A Vector defaults to doubling size of its array**.** While when you insert an element into the ArrayList, it increases its Array size by 50%  **.**

Notes: By default ArrayList size is 10 . It checks whether it reaches the last element then it will create the new array ,copy the new data of last array to new array ,then old array is garbage collected by the Java Virtual Machine (JVM).

**4. Set Increment Size**

Vector defines the increment size and could set the size using setSize method.

public synchronized void setSize(int i) { //some code  }

While there is no setSize() method or any other method in ArrayList which can manually set the increment size.

**5. Enumerator**

Other than Hashtable ,Vector is the only other class which uses both [Enumeration and Iterator](http://javahungry.blogspot.com/2013/06/difference-between-iterator-and-enumeration-collections-java-interview-question-with-example.html) .While ArrayList can only use Iterator for traversing an ArrayList .

**Comparable V.S Comparator Interface(Collection)**

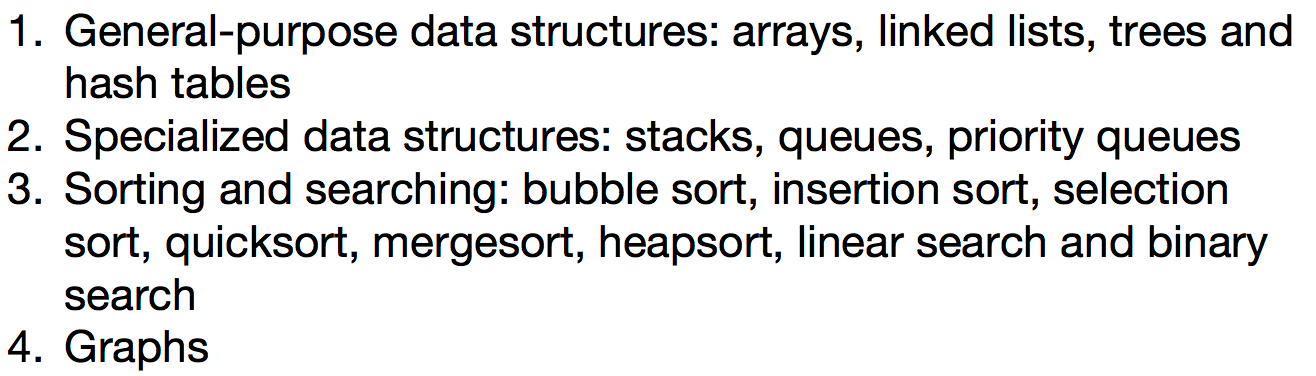
url: <http://javahungry.blogspot.com/2013/08/difference-between-comparable-and.html>

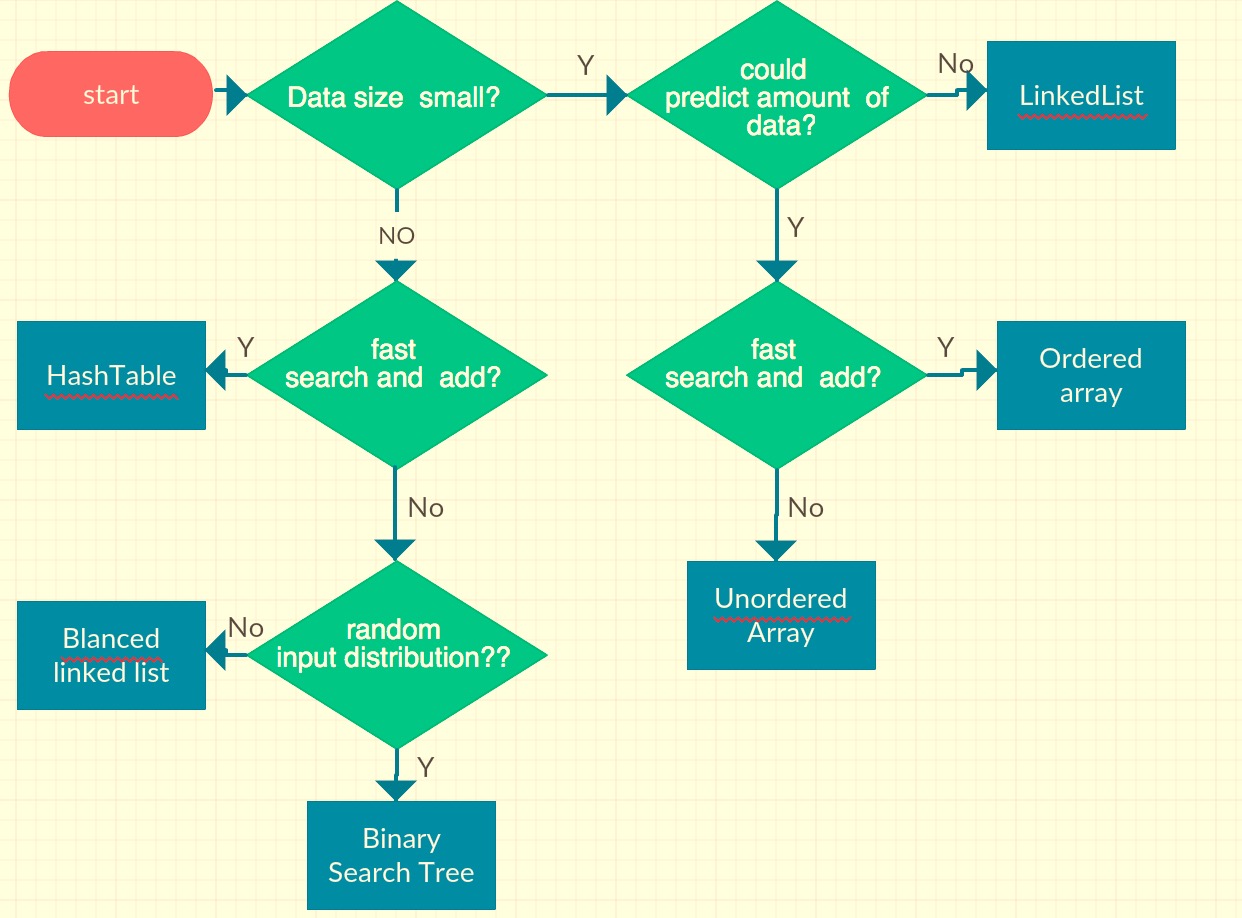
**Iterator V.S Enumeration: Collection Question**

url: <http://javahungry.blogspot.com/2013/06/difference-between-iterator-and-enumeration-collections-java-interview-question-with-example.html>

**Java comman data strctures:**

**(见文件夹中的博客)**





***General data structure:***

(1). Integer:

(2). Array

(3). List: ArrayList and LinkedList:

(4). Hashing: HashTable and HashSet.

(5). Tree: