Java Notes

Logical and Bitwise - bitwise always checks both conditions, eg & checks second even if first is false similarly true in |

Strings
The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.
String, StringBuffer and StringBuilder classes
same string constants have same references
charAt(i)
length()
subString(s,e) [s,e)
contains(ch)
join(str1,delim,str2)
equals(str)
concat(str)
replace(old, new)
split(regex)
indexOf(ch)
compareTo(str) 1-2

String buffer - synchronized - mutable(16) - capacity inc by (oldcapacity*2)+2.
StringBuffer sb=new StringBuffer("Hello ");
capacity()
append (str)
insert(off,str)
replace (s,e,str)
delete (s,e)
reverse ()
length ()
charAt(i)
substring(s,e)

String Builder - non-synchronized - mutable(16) - capacity inc by (oldcapacity*2)+2;
StringBuilder sb=new StringBuilder("Hello ");
same as buffer

StringTokenizer - allows you to break a string into tokens

Array
Size Limit hence use collection
can use for each

System.arraycopy(copyFrom, startS, copyTo, startD, no);
Wrapper Class

to convert primitive into object and object into primitive.

Stack - LIFO, The Stack class extends Vector which implements the List interface. A Vector is a re-sizable collection.
Stack<Type> stack = new Stack<Type>();
push(obj)
pop() -> obj / error
peek() -> obj
size() -> int
empty() -> boolean
search (obj) -> index / -1

Queue - FIFO, The Queue interface extends the Collection interface.
Queue<Type> q = new LinkedList<Type>();
add(obj) -> throws exception
offer(obj) -> return balue
remove() -> obj / error
poll() -> obj / null
peek() -> obj
empty() -> boolean
search (obj) -> index / -1

Deque - Double Ended Queue
Deque<String> deque = new LinkedList<String>();
iterator(): Returna an iterator for this deque.
descendingIterator(): Returns an iterator that has the reverse order for this deque.
use iterator.hasNext()

Priority Queue - processed based on the priority,comparator,doesn’t permit null, default initial capacity (11)
PriorityQueue<obj> pQueue = new PriorityQueue<obj>();
add(obj)
offer(obj)
remove() -> obj / null
poll() -> obj / error
peek()
contains() -> boolean
clear()
size() -> int
toArray()

LinkedList - linked using pointers and addresses, O(1) insert and delete, O(n) access
LinkedList<obj> object = new LinkedList<obj>();
add(obj), add(obj, index), addFirst(obj), addLast(obj) -> same with offer()
get(index), getFirst(), getLast() -> same with poll()
clear()
clone()
contains(obj) -> boolean
peek(), peekFirst(), peekLast()
indexOf(obj), lastIndexOf(obj)
set(index, obj)
size()
toArray()
splIterator() - returns a Spliterator which is late-binding and fail-fast with the same elements as LinkedList.

Heaps - A Heap is a special Tree-based data structure in which the tree is a complete binary tree.
max heap - max on top
min heap - min on top
heap represented as array - Arr[(i-1)/2] - parent node
Arr[(2*i)+1] - left child node
Arr[(2*i)+2] - right child node

```
PriorityQueue<obj> minHeap = new PriorityQueue<obj>();
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(new Comparator<Integer>(){
        public int compare(int a, int b){
                return -1 * a.compareTo(b);
        }
});
```

Rest as PriorityQueue
See heapify