List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List
CopyOnWriteArrayList	O(n)	O(n)	O(1)	O(n)	O(1)	Array

Set	Add	Remove	Contains	Next	Size	Data Structure
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
EnumSet	O(1)	O(1)	O(1)	O(1)	O(1)	Bit Vector
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Red-black tree
CopyOnWriteArraySet	O(n)	O(n)	O(n)	O(1)	O(1)	Array
ConcurrentSkipListSet	O(log n)	O(log n)	O(log n)	O(1)	O(n)	Skip List

Queue	Offer	Peak	Poll	Remove	Size	Data Structure
PriorityQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedList	O(1)	O(1)	O(1)	O(1)	O(1)	Array
ArrayDequeue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)	O(n)	Linked List
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Array
PrioriritыBlockingQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
SynchronousQueue	O(1)	O(1)	O(1)	O(n)	O(1)	None!
DelayQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List

Map	Get	ContainsKey	Next	Data Structure
HashMap	O(1)	O(1)	O(h / n)	Hash Table
LinkedHashMap	O(1)	O(1)	O(1)	Hash Table + Linked List
IdentityHashMap	O(1)	O(1)	O(h / n)	Array
WeakHashMap	O(1)	O(1)	O(h / n)	Hash Table
EnumMap	O(1)	O(1)	O(1)	Array
TreeMap	O(log n)	O(log n)	O(log n)	Red-black tree
ConcurrentHashMap	O(1)	O(1)	O(h / n)	Hash Tables
ConcurrentSkipListMap	O(log n)	O(log n)	O(1)	Skip List

Casting, Conversion

```
int x = (int)5.5; //works for numeric types
int x = Integer.parseInt("123");
float y = Float.parseFloat("1.5");
int x = Integer.parseInt("7A",16); //fromHex
String hex = Integer.toString(99,16);//toHex
```

Number types

```
Integer x = 5; double y = x.doubleValue();
double y = (double)x.intValue();
```

String Methods

```
//Operator +, e.g. "fat"+"cat" -> "fatcat"
boolean equals(String other);
int length();
char charAt(int i);
String substring(int i, int j); //j not incl
boolean contains(String sub);
boolean startsWith(String pre);
boolean endsWith(String post);
int indexOf(String p); //-1 if not found
int indexOf(String p, int i); //start at i
int compareTo(String t);
String replaceAll(String str, String find);
String[] split(String delim);
```

Math

```
Math.abs(NUM),Math.ceil(NUM),Math.floor(NUM)
Math.log(NUM),Math.max(A,B),Math.min(C,D),
Math.pow(A,B),Math.round(A),Math.random()
```

ARRAYS:

```
int[] x = new int[10]; //ten zeros
int[][] x = new int[5][5]; //5 by 5 matrix
int[] x = {1,2,3,4};
x.length; //int expression length of array
int[][] x = {{1,2},{3,4,5}}; //ragged array
String[] y = new String[10]; //10 nulls
```

List<T> ArrayList<T>: Slow insert into middle
LinkedList<T>: slow random access
Stack: Removes and adds from end

List Usage:

```
boolean add(T e);
void clear(); //empties
boolean contains(Object o);
T get(int index);
T remove(int index);
boolean remove(Object o);
T set(int index, E val);
Int size();
```

Queue<T> LinkedList implements Queue

Queue Usage:

```
T element(); // does not remove
boolean offer(T o); //adds
T peek(); //pike element
T poll(); //removes
T remove(); //like poll
```

Set<T>: uses Comparable<T> for uniqueness
TreeSet<T>, items are sorted
HashSet<T>, not sorted, no order
LinkedHashSet<T>, ordered by insert
Usage like list: add, remove, size

Map<K,V>: Pairs where keys are unique
HashMap<K,V>, no order
LinkedHashMap<K,V> ordered by insert
TreeMap<K,V> sorted by keys

```
V get(K key);
Set<K> keySet(); //set of keys
V put(K key, V value);
V remove(K key);
Int size();
Collection<V> values(); //all values
```

java.util.Collections

```
Collections.sort(x); //sorts with comparator
Collections.max( … ); //returns maximum
Collections.min( … ); //returns maximum
Collections.copy( A, B); //A list into B
Collections.reverse( A ); //if A is list
```

Sort Using Comparator:

```
Collections.sort(x, new Comparator<T>{
  public int compareTo(T a, T b) {
    //calculate which is first
    //return -1, 0, or 1 for order:
    return someint;
  }
}
```

```
List<Integer> list = new ArrayList<Integer>();
Queue<Double> queue = new LinkedList<Double>();
Stack<String> stack = new Stack<String>();
Set<String> words = new HashSet<String>();
Map<String, Integer> map = new TreeMap<>();
```

Lists, Stacks, Queues, Sets, Maps

clear()
equals(**collection**)
isEmpty()
size()
toString()

ArrayList, LinkedList, HashSet, TreeSet

add(**value**)
contains(**value**)
remove(**value**)
removeAll(**collection**)
retainAll(**collection**)

List<E>

add(**index**, **value**)
indexOf(**value**)
get(**index**)
lastIndexOf(**value**)
remove(**index**)
set(**index**, **value**)
subList(**from**, **to**)

Stack<E>	Queue<E>
peek()	add(**value**)
pop()	peek()
push(**value**)	remove()

Map<K, V>

containsKey(**key**)
get(**key**)
keySet()
put(**key**, **value**)
putAll(**map**)
remove(**key**)
toString()
values()