# CECS 277 – Lecture 8 – Collections

Collections are containers are used to store groups of elements, in a single object. Collections expand and contract as new elements are added and removed. Each type of collection stores elements in different ways and have methods to add, access, and remove elements. Choosing one depends on the needs of the program.

**List –** A list is used when the elements need to be in an ordered sequence, or indexing is needed. A list is capable of storing duplicate values and elements may be added or removed at any position in the list. (ex: Alphabetical list of names).

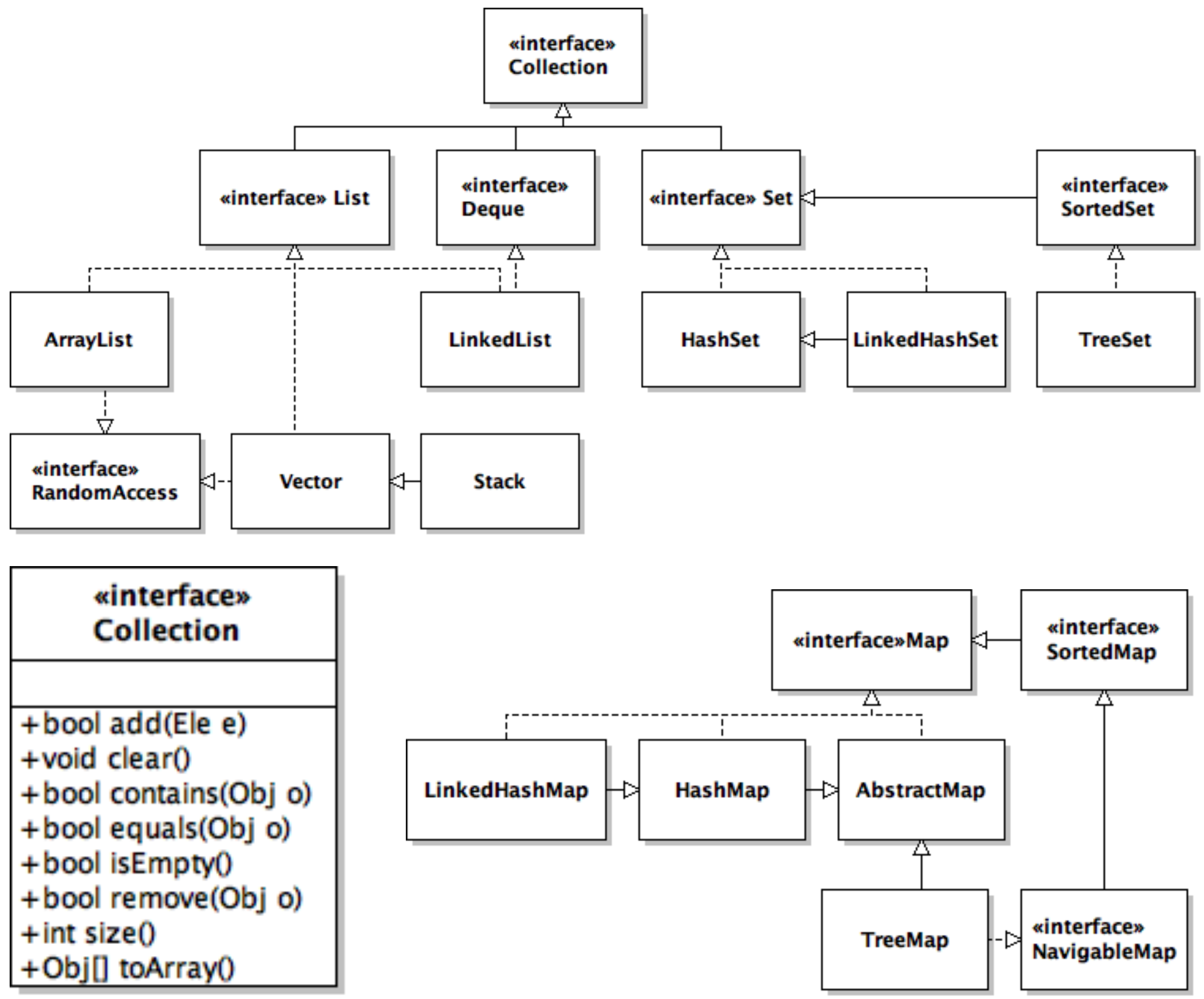
**Set –** A set provides an unordered collection of unique elements. It is not indexable and it does not store duplicate values. (ex: Names of students in a class).

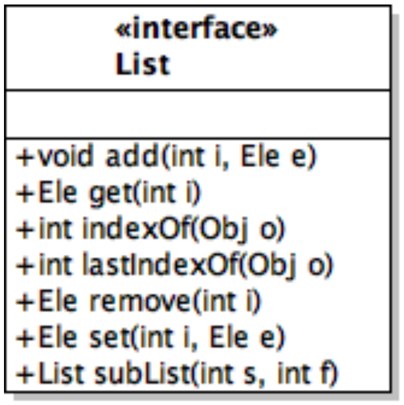
**Queue –** A queue stores elements for processing, used in FIFO order. (ex: Songs in a playlist).

**Deque –** Double-Ended Queue – A deque stores elements for processing. Can be used both as FIFO and LIFO.

**Map –** A map stores key-value pairs. Each key maps to a single value. There cannot be any duplicate keys in a map, but values may be duplicated. (ex: States and their populations).

**Figure**: The class hierarchy for Java's Collections.





**Lists –** Lists change their size as items are added and removed from the collection. Elements are added and removed anywhere in the list by specifying an index. Elements in the list are stored in inserted or sorted order and they may be accessed randomly. Classes that implement List: ArrayList, LinkedList, Vector, and Stack. All of the methods for each of these classes can be found in the JavaAPI.

**ArrayList** – good for fast random indexing, but it can often be slower when inserting and removing many items since it needs to expand and contract the list. It should not be used in multi-threaded programs.

```
ArrayList<Animal> animals = new ArrayList<Animal>();
animals.add( new Cat( “Fluffy” ) );
animals.add( new Dog( “Spot” ) );
int size = animals.size();
for( Animal a : animals ){
      System.out.println( a.getName() );
}
```

**LinkedList** – is made up of a series of connected nodes arranged in a doubly-linked list. This makes it very fast when adding and removing from the list, especially at the front or end, but slower when trying to access values in the list. It should not be used in multi-threaded programs.

```
LinkedList<Animal> animals = new LinkedList<Animal>();
animals.addLast( new Cat( “Fluffy” ) );
animals.addFirst( new Dog( “Spot” ) );
Animal c = animals.getFirst();
Animal d = animals.getLast();
animals.removeLast();
```

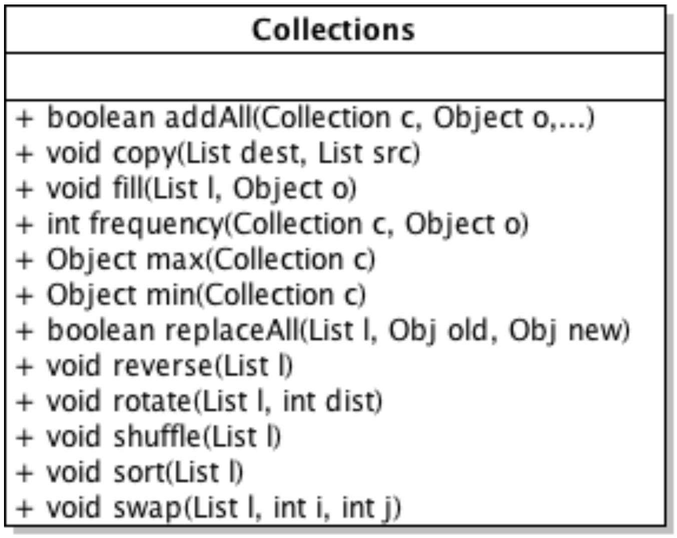
**Vector** – similar to ArrayList, but it is safe for multi-threading.

```
Vector<Animal> animals = new Vector<Animal>();
animals.add( new Cat( “Fluffy” ) );
animals.add( new Dog( “Spot” ) );
Animal d = animals.get(1);
Animal c = animals.remove(0);
animals.clear();
```

**Stack** – derived from Vector, but implements Last-In First-Out (LIFO) behavior. Has normal stack functions: push(), pop(), peek(), rather than add(), remove(), and get().

```
Stack<Animal> animals = new Stack<Animal>();
animals.push( new Cat( “Fluffy” ) );
animals.push( new Dog( “Spot” ) );
Animal a = animals.peek();
animals.pop();
```

**Collections Class –** is a set of static methods that operate on collections. The following is a short list of examples of some commonly used methods. A full list can be found in the Java API: http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html



**Collections**

+ boolean addAll(Collection c, Object o,...)
+ void copy(List dest, List src)
+ void fill(List l, Object o)
+ int frequency(Collection c, Object o)
+ Object max(Collection c)
+ Object min(Collection c)
+ boolean replaceAll(List l, Obj old, Obj new)
+ void reverse(List l)
+ void rotate(List l, int dist)
+ void shuffle(List l)
+ void sort(List l)
+ void swap(List l, int i, int j)

Some of these methods work specifically on a particular type of collection, such as a list or a set, other methods work on any type of collection.

To use the Collections class, import the java.util library.

**Example**:

```
import java.util.Collections;
import java.util.ArrayList;
public class CollectionTest {
     public static void main( String []args ) {
          ArrayList<Integer> list = new ArrayList<Integer>();

          Collections.addAll( list, 3, 2, 5, 6, 1, 9, 4, 7, 8, 0 );

          System.out.println(list); //[3, 2, 5, 6, 1, 9, 4, 7, 8, 0]

          Collections.sort(list);
          System.out.println(list); //[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

          Collections.reverse(list);
          System.out.println(list); //[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

          Collections.shuffle(list);
          System.out.println(list); //[4, 6, 8, 3, 1, 9, 7, 0, 5, 2]

          Collections.rotate(list, 3);
          System.out.println(list); //[0, 5, 2, 4, 6, 8, 3, 1, 9, 7]

          Collections.swap(list, 4, 5);
          System.out.println(list); //[0, 5, 2, 4, 8, 6, 3, 1, 9, 7]

          int min = Collections.min(list);
          int max = Collections.max(list);
          System.out.println(“Min=” + min + “ Max=” + max); //0, 9

          Collections.replaceAll(list, 3, 1);

          int freq = Collections.frequency(list, 1);
          System.out.println(“Frequency of value 1 =”+ freq); //2

          ArrayList<Integer> list2 = new ArrayList<Integer>();
          Collections.addAll(list2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1);
          Collections.copy(list2, list);

          System.out.println(list2); //[0, 5, 2, 4, 8, 6, 1, 1, 9, 7]

          Collections.fill(list, 0);
          System.out.println(list); //[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
     }
}
```