ArrayList

Colorado State University

Department of Computer Science

Slides Originally Created by Albert Lionelle (Albert.Lionelle@colostate.edu),
updated by Marcia Moraes (marcia.moraes@colostate.edu)

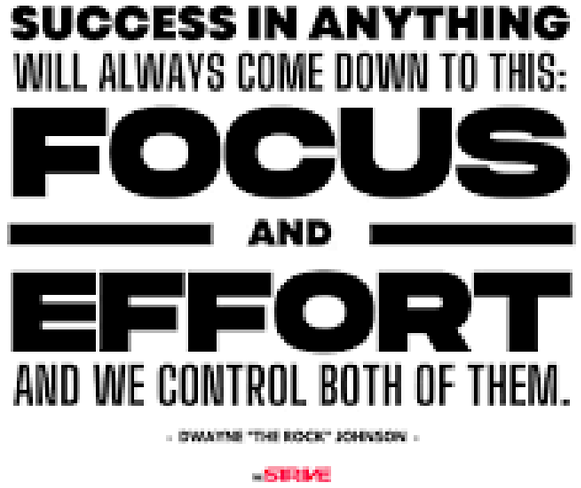
Announcements

TODO Reminders:

Readings are due **before** lecture

- Reading 11 (zybooks) – you should have already done that ☺
- Lab 07
- Reading 12 (zyBooks)
- Lab 08
- Reading 13 (zybooks)
- RPA 6

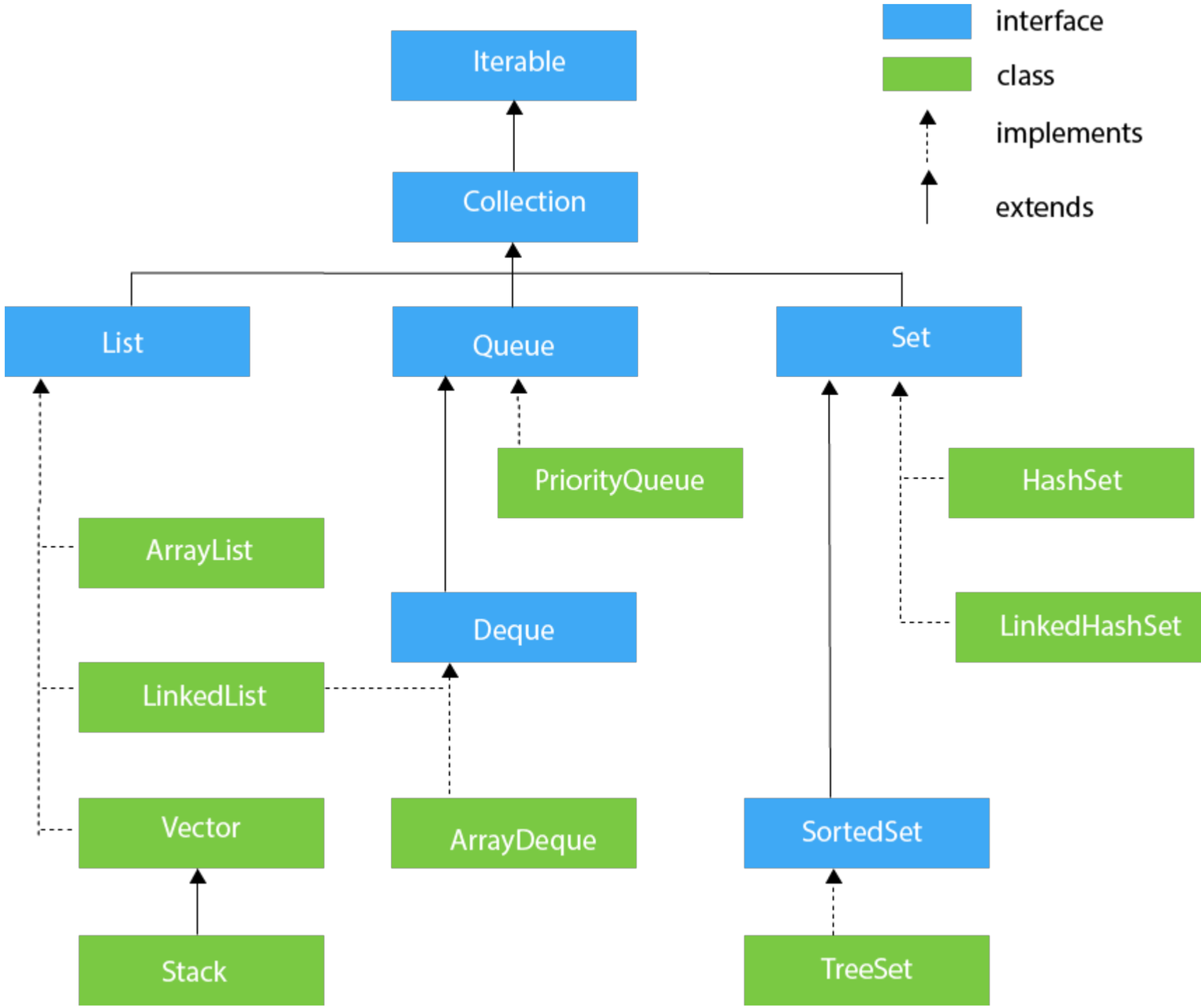
Keep practicing your RPAs in a spaced and mixed manner ☺



https://thestrive.co/100-best-effort-quotes-to-help-you-win-in-life/

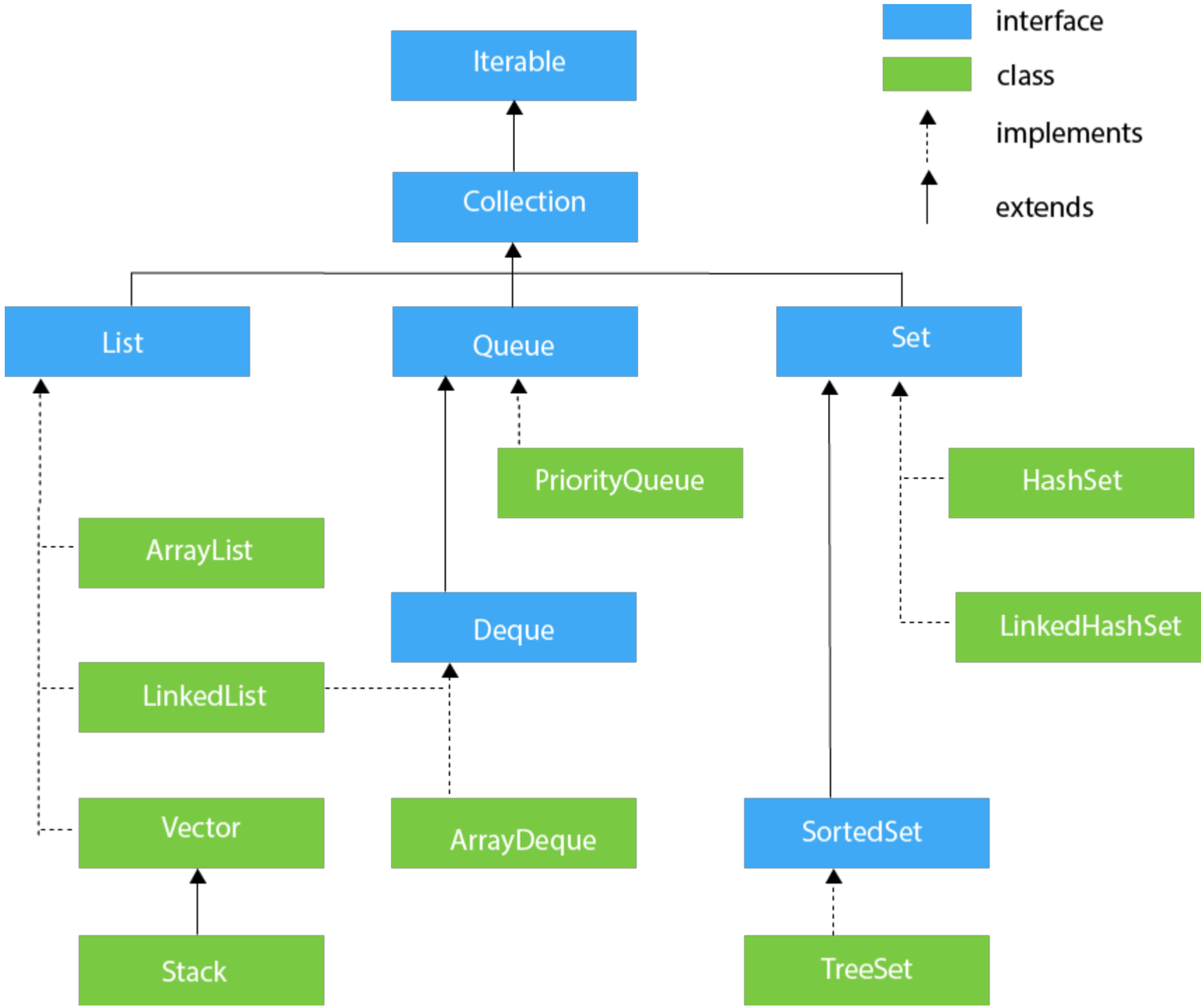
Collections

- A collection represents a group of objects, known as its *elements*.
- Some collections allow duplicate elements and others do not. Some are ordered and others unordered.
- It is an Interface
  - We will learn more later
  - For now, an interface specifies a set of behaviors (methods) that other classes needs to implement.



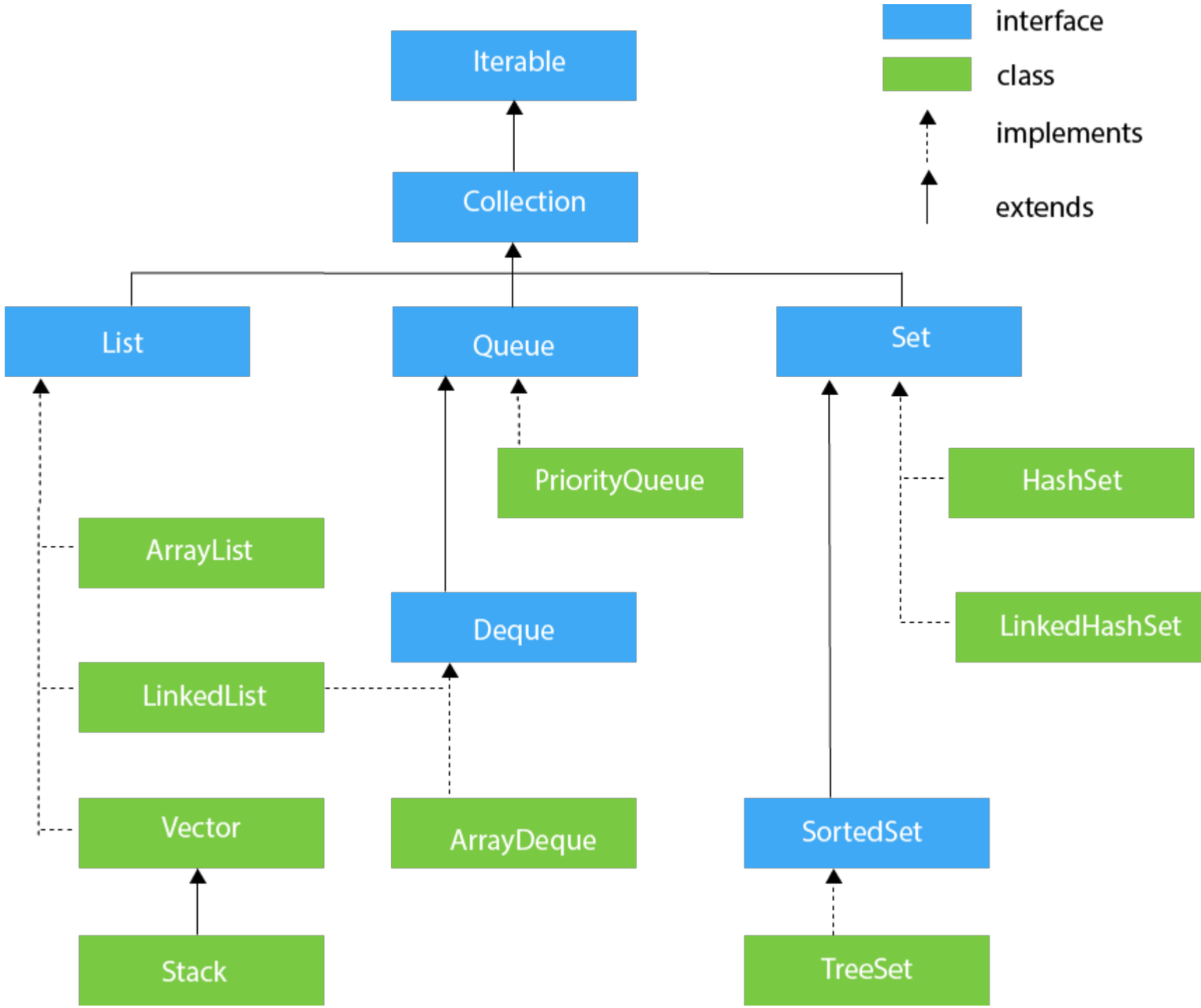
List

- An ordered collection (also known as a *sequence*).
- The user of this interface has precise control over where in the list each element is inserted.
- The user can access elements by their integer index (position in the list), and search for elements in the list.

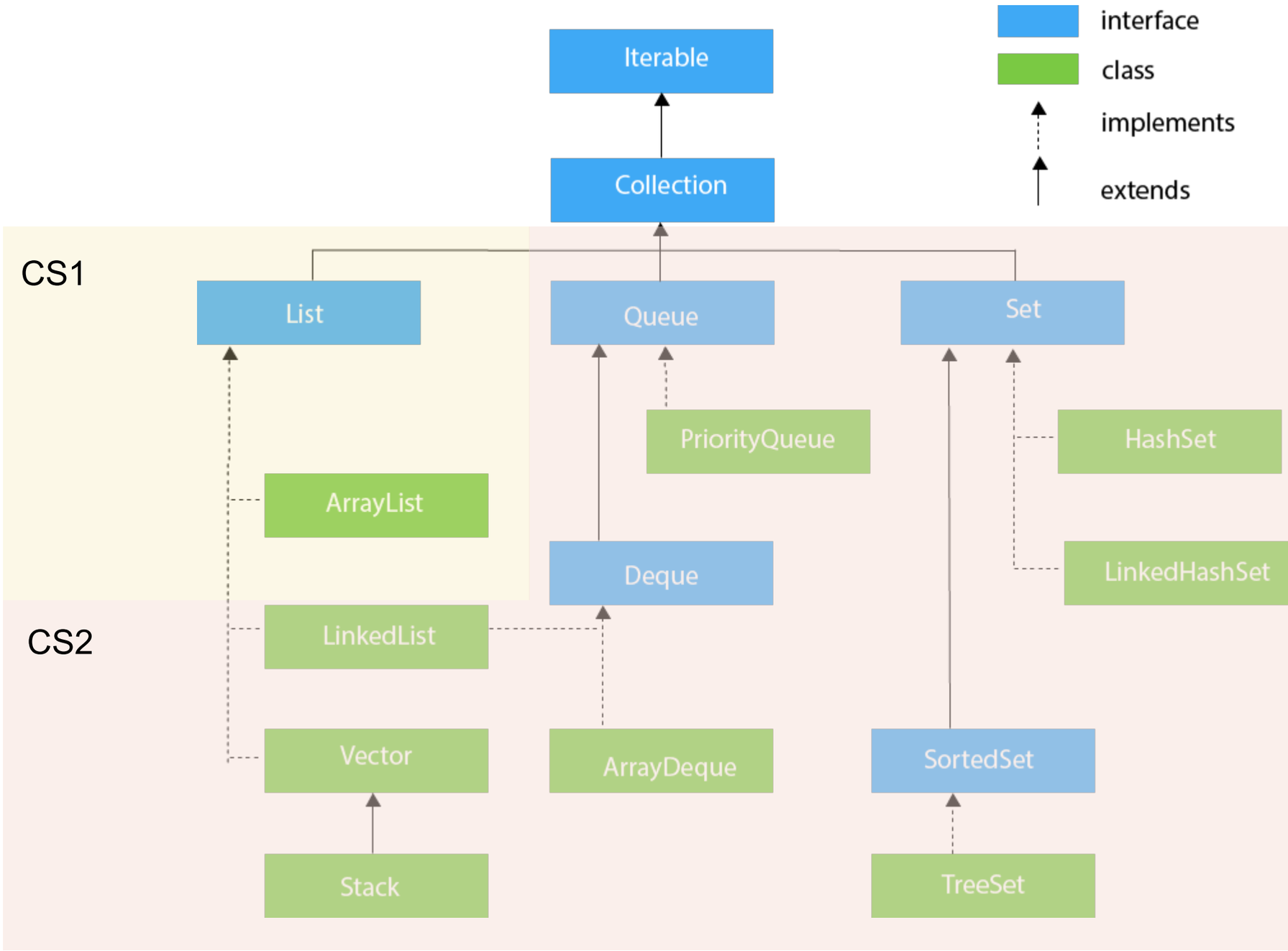


ArrayList

- Resizable-array implementation of the List Interface.
- Each ArrayList instance has a capacity.
- Capacity is the size of the array used to store the elements.
- When an instance of an ArrayList reaches its capacity, the instance grown automatically (resize), by doubling its initial capacity.

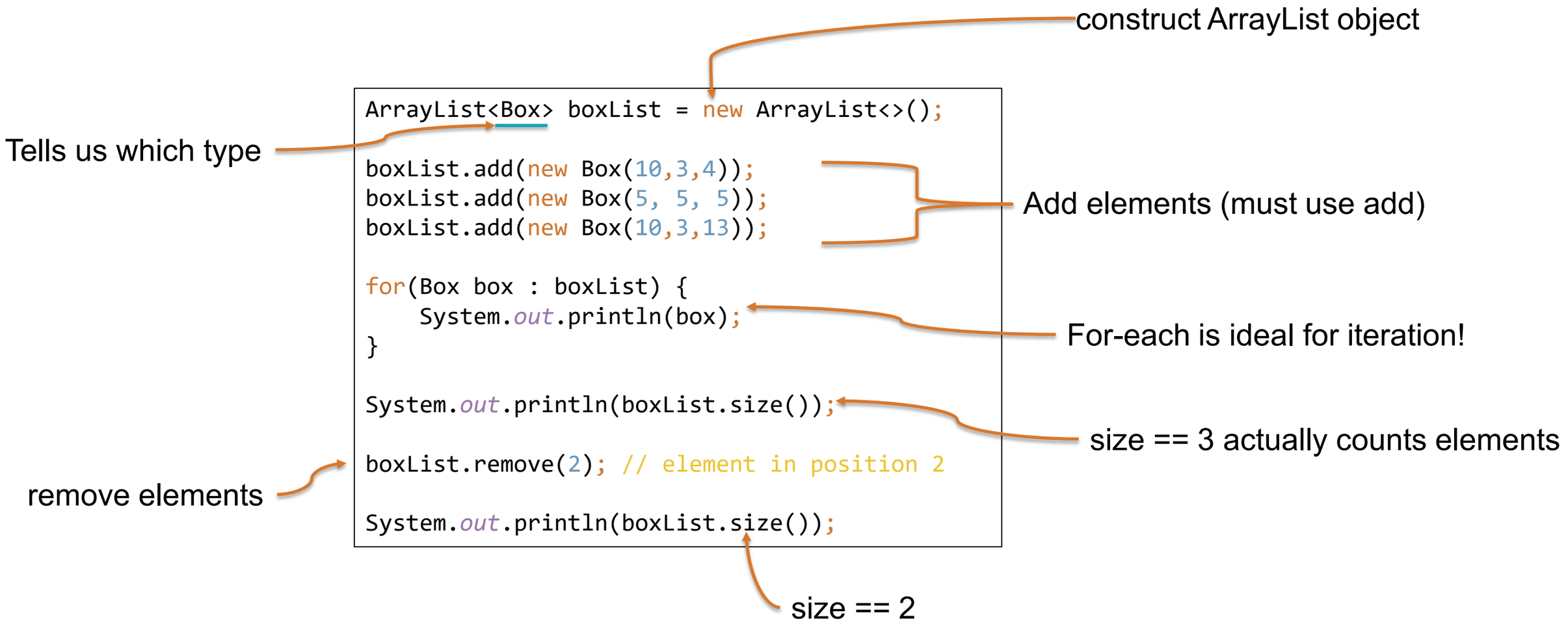


Collections
CS1 and CS2



ArrayList in Code

```
public class Box {
    private int width;
    private int height;
    private int length;
    /** getters and setters **/
    public int getArea() {
        return getHeight()*getLength()*getWidth();
    }
 //…
}
```



ArrayList in Code

```
ArrayList<Box> boxList = new ArrayList<>();
```

Creates a space to store 10 Boxes, all elements are null.

Memory Stack

variable	value
boxList	ArrayList@7ba7

ArrayList@7ba7

index	object
0	null
1	null
2	null
3	null
4	null
5	null
6	null
7	null
8	null
9	null

ArrayList in Code

```
boxList.add(new Box(10,3,4));
boxList.add(new Box(5, 5, 5));
boxList.add(new Box(10,3,13));
```

```
public class Box {
    private int width;
    private int height;
    private int length;
    /** getters and setters **/
    public int getArea() {
        return getHeight()*getLength()*getWidth();
    }
 //…
}
```

Memory Stack

variable	value
boxList	ArrayList@7ba7

ArrayList@7ba7

index	object
0	Box@01
1	Box@02
2	Box@03
3	null
4	null
5	null
6	null
7	null
8	null
9	null

Box@01	
width	10
height	3
length	4

Box@02	
width	5
height	5
length	5

Box@03	
width	10
height	3
length	13

Wrapper Classes

- ArrayList
  - Only Stores Objects
- Wrapper Classes to the rescue
  - int has Integer
  - double has Double
  - boolean has Boolean
  - char has Character
  - (and so on)
- Boxing and Unboxing
  - Allows automatic conversion between wrapper and primitive
  - Integer myInt = 10;
    - Same as Integer myInt = new Integer(10);

```
ArrayList<Integer> list = new ArrayList<>();

list.add(10);
list.add(2);
list.add(-1);

int total = list.get(0) + list.get(1);
System.out.println(total); // prints 12
```

```
A simplest representation:
[10, 2, -1]

That is actually how it is going to be printed
if you print the ArrayList reference.
```

ArrayList: Some Methods

- `.get(int)` - returns the item at the set index
- `.remove(int)` - removes the item at the set index (giving the item if needed)
- `.set(int, value)` - replaces an item at an already exist index
- `.add(value)` - adds an item to the end of the list
- `.add(int, value)` - inserts an item at a set location
- `.size()` - returns the total number of elements in the list
  - yes, this one is confused with .length() all the time.

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

For Each

```
for(Type element: list)
```

- Get all the elements that are in a list starting from the begin until the end

```
ArrayList<Integer> list = new ArrayList<>();

list.add(10);
list.add(2);
list.add(-1);

for(Integer element: list)
    System.out.println(element);
```

Example

```
import java.util.ArrayList;
public class ArrayListInts {
   public static void main(String args[]){
      //creates an Arraylist named lst with 10 initial capacity
      ArrayList<Integer> lst = new ArrayList<>();
      lst.add(11);
      lst.add(20);
      lst.add(40);
      lst.add(33);
      System.out.println("Size of the list: " + lst.size());
      for(int i = 0; i < lst.size(); i++){
         System.out.println("Index: " + i + " Element: " + lst.get(i));
      }
      System.out.println("Remove element from index 2");
      lst.remove(2);
      System.out.println("Size of the list: " + lst.size());
      for(int i = 0; i < lst.size(); i++){
         System.out.println("Index: " + i + " Element: " + lst.get(i));
      }
      System.out.println("Add 1 in index 0");
      lst.add(0, 1);
      System.out.println("Size of the list: " + lst.size());
      for(Integer elem: lst){
         System.out.println(elem);
      }
      System.out.println("Remove 1 from the list");
      lst.remove((Integer)1);
      System.out.println("Size of the list: " + lst.size());
      for(int i = 0; i < lst.size(); i++){
         System.out.println("Index: " + i + " Element: " +
lst.get(i));
      }
   }
}
```

What is the difference between `lst.remove((Integer)1);` and `lst.remove(1);` instructions?