A Collection

A ***collection*** is just an object that represents a group of objects.

In general, the group of objects have some relationship to each other.

Computer science has common names, and an expected set of behavior, for different types of collection objects.

Collection objects, in various languages, include arrays, lists, vectors, sets, queues, tables, dictionaries, and maps.

These are differentiated by the way they store the objects in memory, how objects are retrieved and ordered, and whether nulls and duplicate entries are permitted.

A Collections Framework

Oracle's Java documentation describes it's collections framework as:

"A unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details.”

If you're interested in reading the Oracle documentation, they have a good overview at the following link

https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html

This was written for JDK-8, but it still applies.

What's in the framework, what's not?

Strictly speaking, arrays and the array utilities in the java.util.Arrays class are not considered part of this framework.

All collection objects implement the Collection interface, with the exception of maps, and I'll explain why in this section.

The Collection Interface

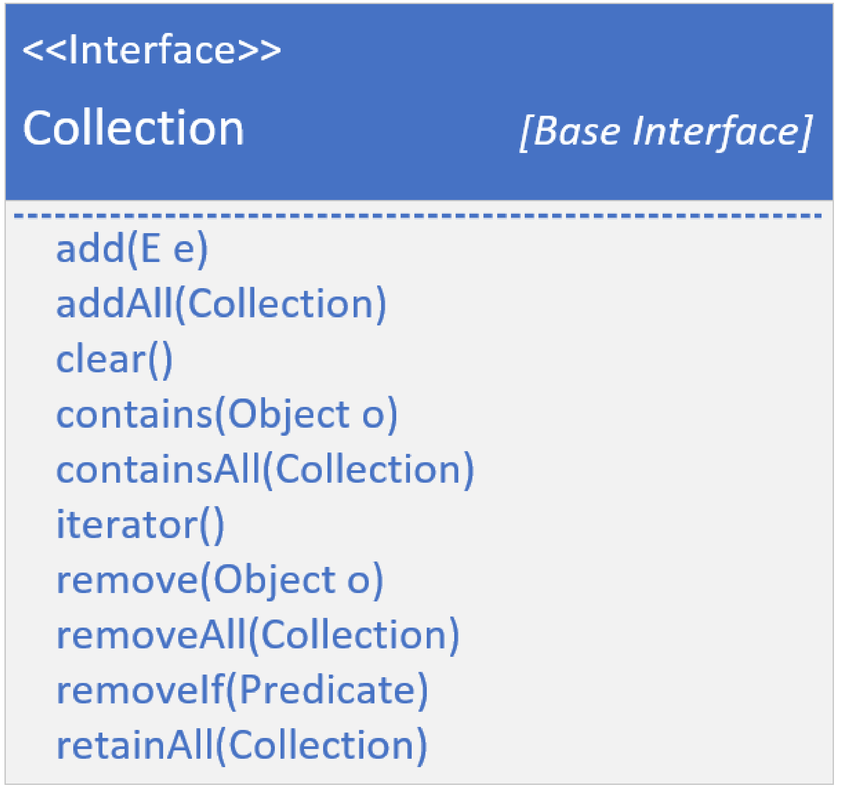
The Collection interface is the root of the collection hierarchy.

Like most roots in software hierarchies, it's an abstract representation of the behavior you'd need, for managing a group of objects.

This type is typically used to **pass collections** around, and manipulate them where **maximum generality** is desired.

Remember, the interface let's us describe objects by what they can do, rather than what they really look like, or how they're ultimately constructed.

If you look at the methods on this interface, you can see the basic operations a collection of any shape, or type, would need to support.



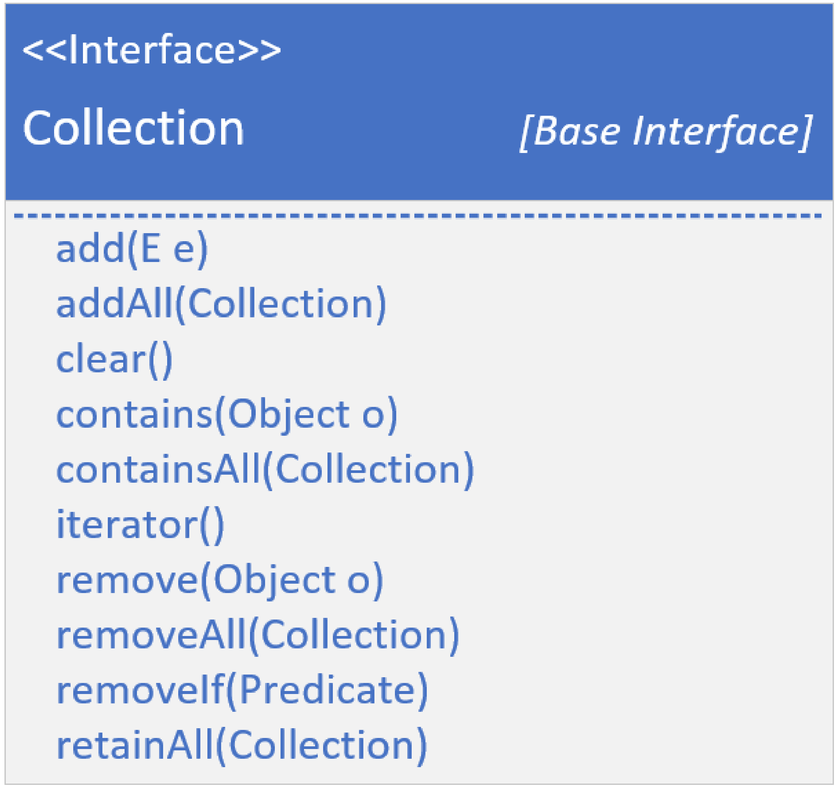
The Collection Interface

We've already looked at every one of these operations, in our study of ArrayLists and LinkedLists.

When managing a group, you'll be adding and removing elements, checking if an element is in the group, and iterating through the elements.

There are some others, but these are the ones that describe nearly everything you'd want to do to manage a group.

Java uses the term **Element** for a member of the group being managed.

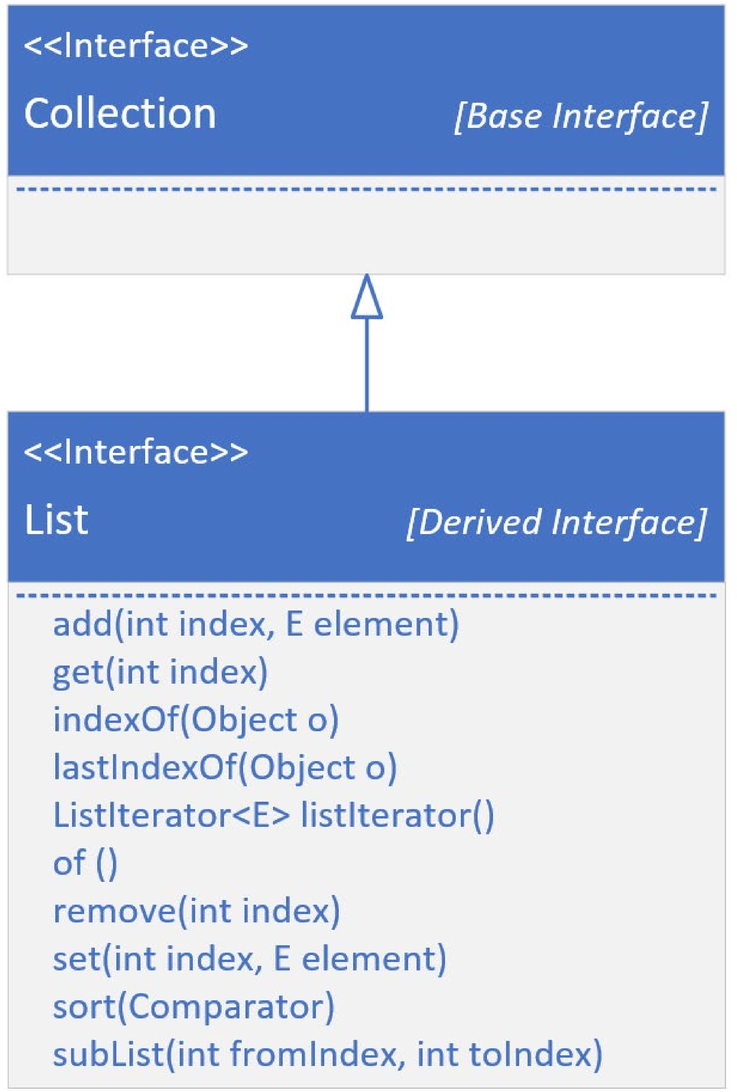


The Collection Interface

This slide shows the List interface extending Collection.

For simplicity, I'm not showing the Collection methods that I showed on the previous slide.

I'm only showing additional methods specifically declared on the List interface.



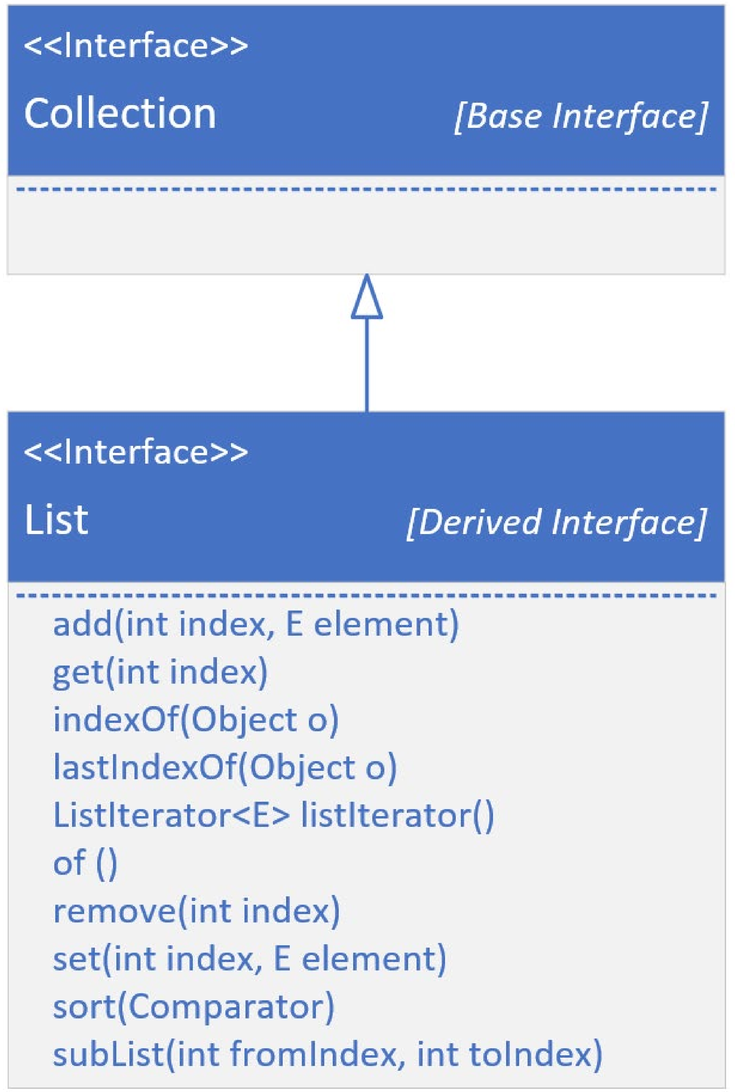
The Collection Interface

We covered most of these methods, but I wanted you to see here, that most of these are dealing with an index.

A list can be either indexed, as an ArrayList, or not, like a LinkedList, but a LinkedList is implemented to support all of these methods as well.

Derived interfaces may have specific ways to add, remove, get, and sort elements for their specific type of collection, in addition to those defined on the Collection Interface itself.

Now, let's look at the big picture of interfaces, and some specific implementations.

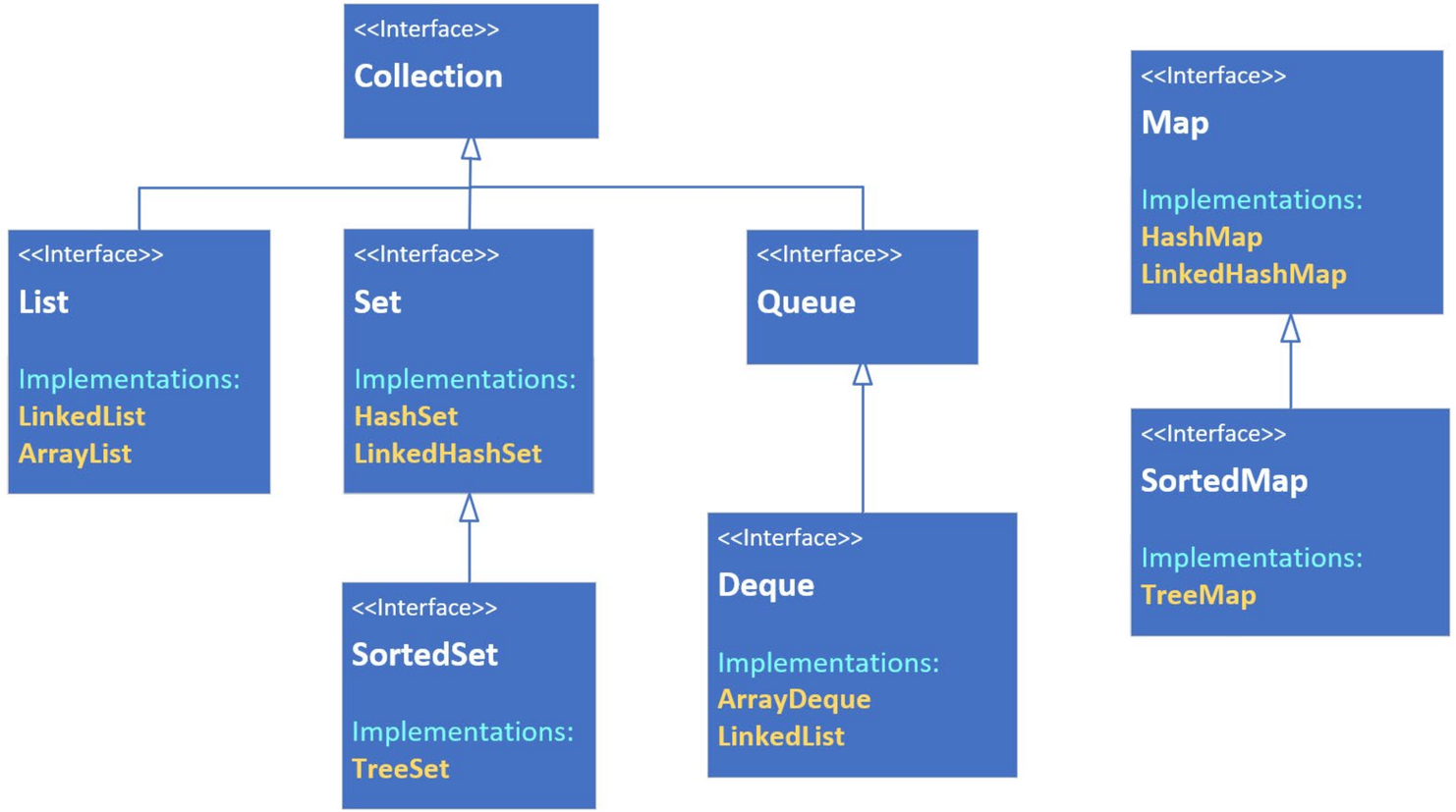


Collections - The Big Picture

This slide shows the interface hierarchy.

It's also showing the implementations or concrete classes, that implement these interfaces, in yellow font.

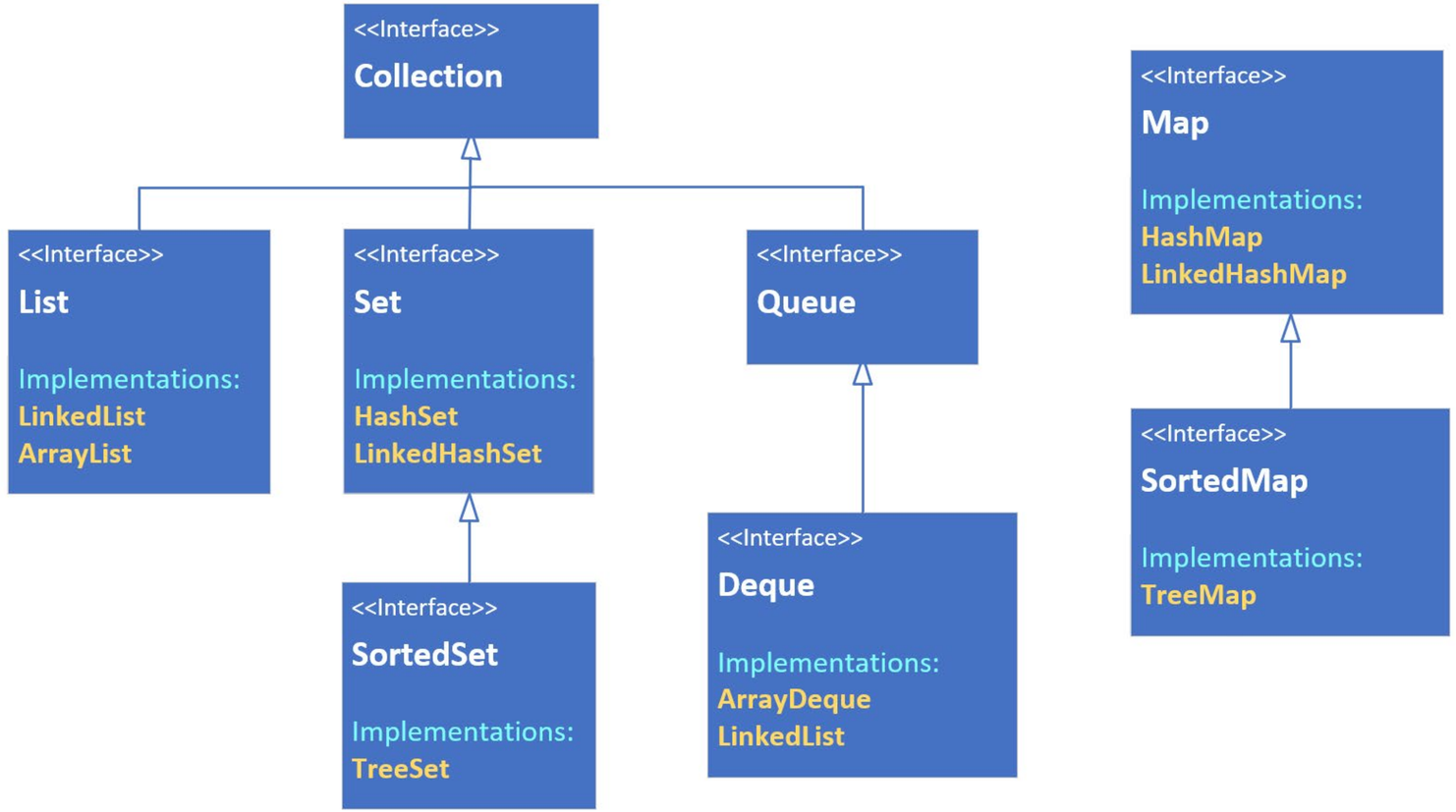
Notice that Map does not extend Collection, although still part of this framework.



Collections - The Big Picture

Maps are uniquely different, which I'll be explaining when I cover Maps in this section.

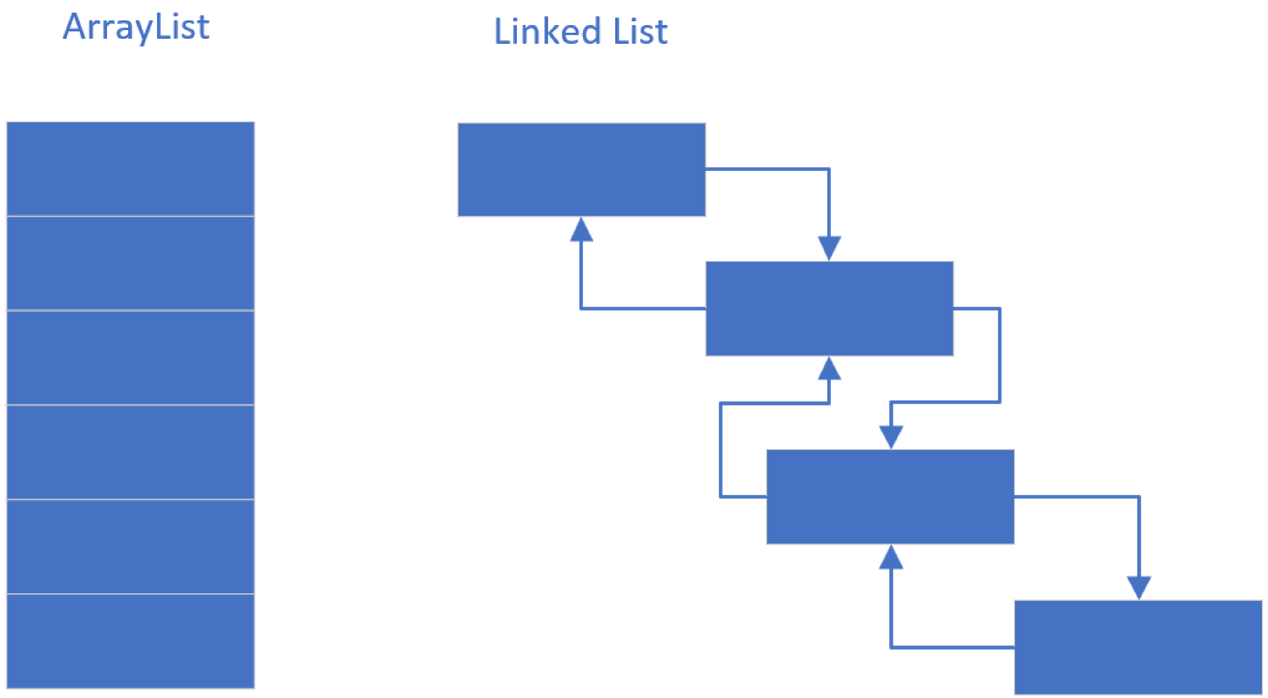
You can see that LinkedList implements both List and Deque, and I discussed this in detail when I covered LinkedLists.



The List

A List is An ordered collection (also known as a *sequence*).

These can be sequenced in memory like an ArrayList, or maintain links to the next and previous values, as a LinkedList.

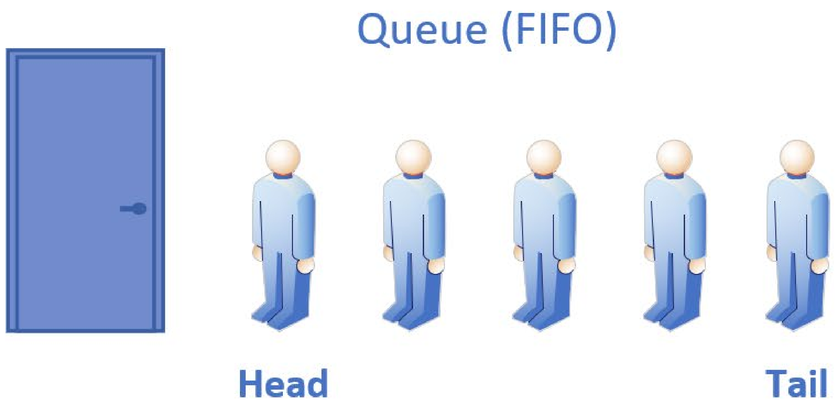


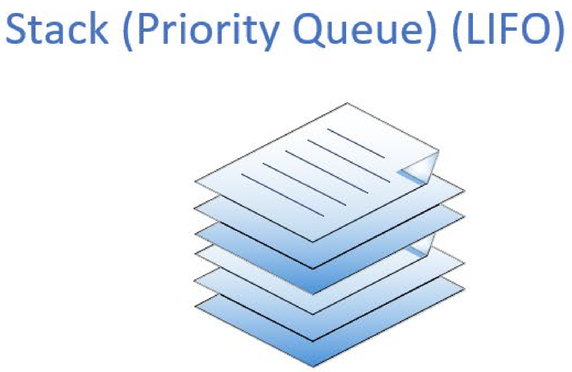
The Queue

A Queue is a collection designed for holding elements prior to processing, in other words the processing order matters, so the first and last positions, or the head and tail, are prioritized.

Most often these may be implemented as First In, First Out (FIFO), but can be implemented like a Stack, as Last In First Out (LIFO) which we've discussed.

Remember a Deque supports both.





The Set

A Set is a collection conceptually based off of a mathematical set.

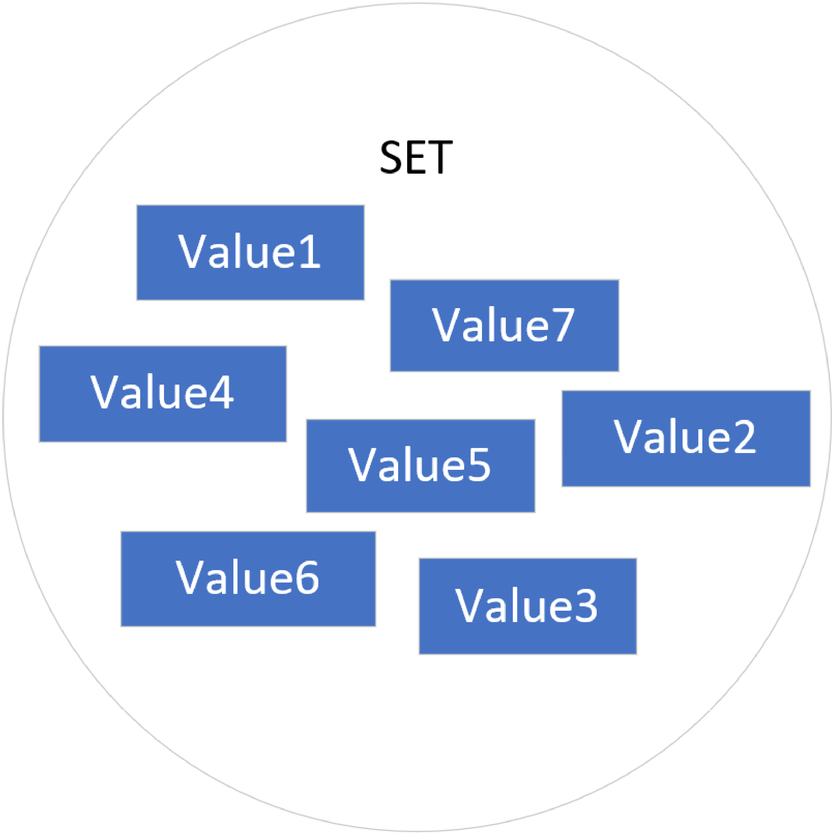
Importantly, it contains **no duplicate elements**, and isn't naturally sequenced or ordered.

You can think of a set as a kind of penned in chaotic grouping of objects.

Java has three implementations, which I'll be reviewing in this section of the course in detail, the HashSet, the TreeSet and the LinkedHashSet.

These are distinguished by the underlying way they store the elements in the set.

A Sorted Set is a set that provides a total ordering of the elements.



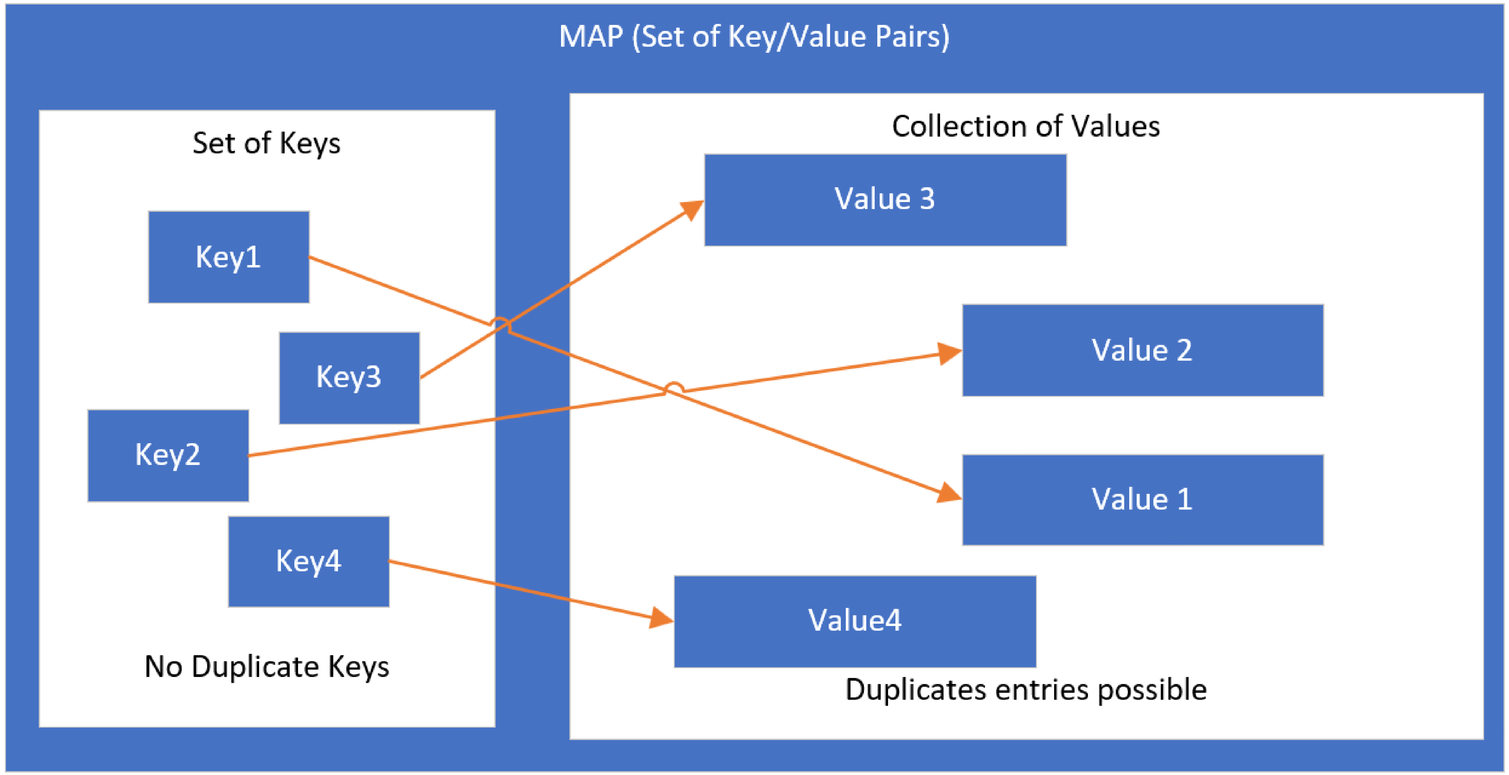
The Map

A Map is a collection that stores key and value pairs.

The keys are a set, and the values are a separate collection, where the key keeps a reference to a value.

Keys need to be unique, but values don't.

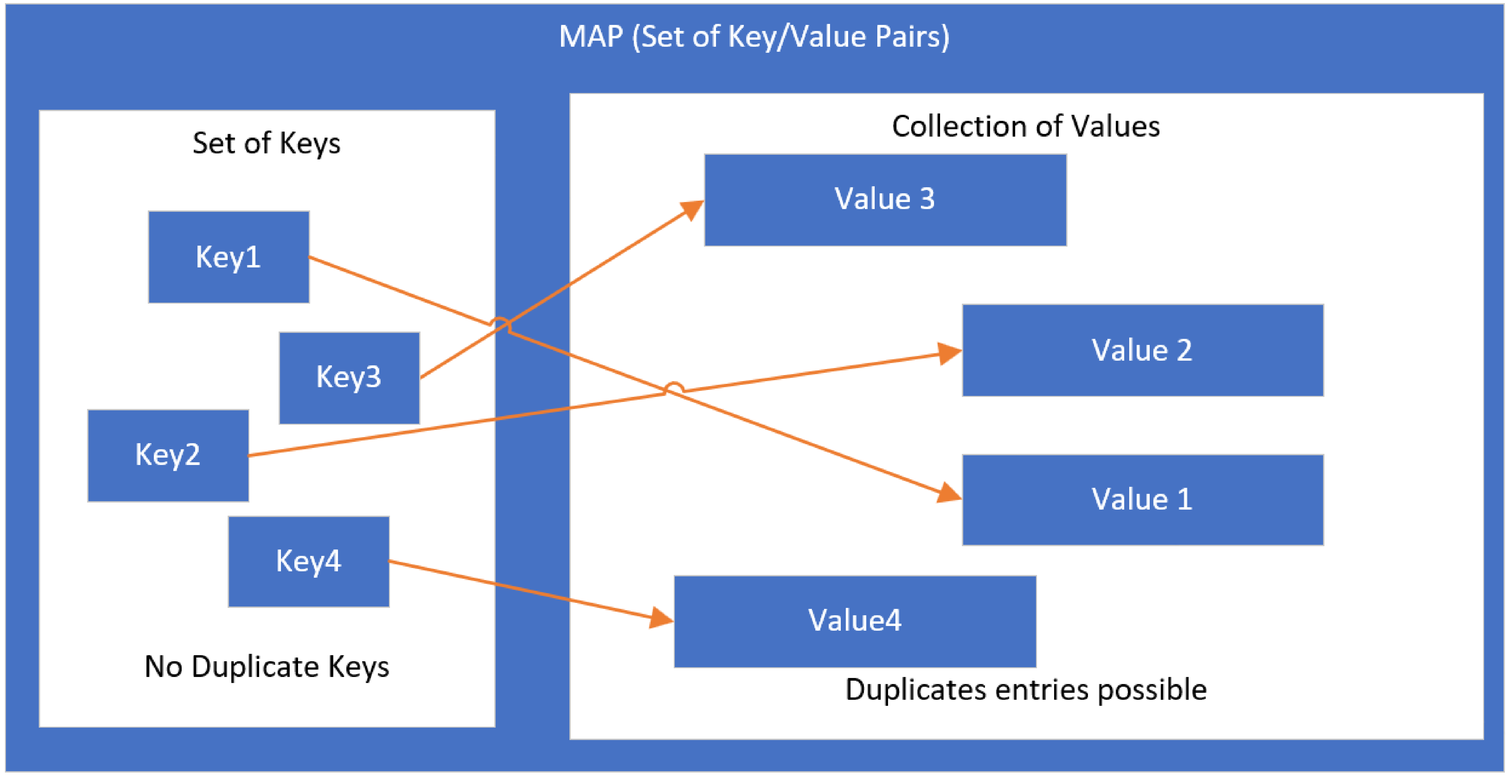
Elements in a tree are stored in a key value Node, also called an Entry.



The Map

In the videos coming up, we'll be looking at Set and Map, and how they resemble and differ from List.

But before that, I want to talk about polymorphic algorithms.



What's a polymorphic algorithm?

Oracle's documentation describes a polymorphic algorithm as a piece of reusable functionality.

At one time, most of these methods were provided to us, as static methods, on a class called **java.util.Collections**.

Since JDK-8, and the advent of multiple interface enhancements, some of these methods are now on the interfaces themselves, as default or static methods.

But not all, so I'll be discussing this class, and what it has to offer, in comparison to what's available on each collection class.

It's also important to understand that legacy code will be using this class for some operations, that can be done from the class itself.