The Collections Framework

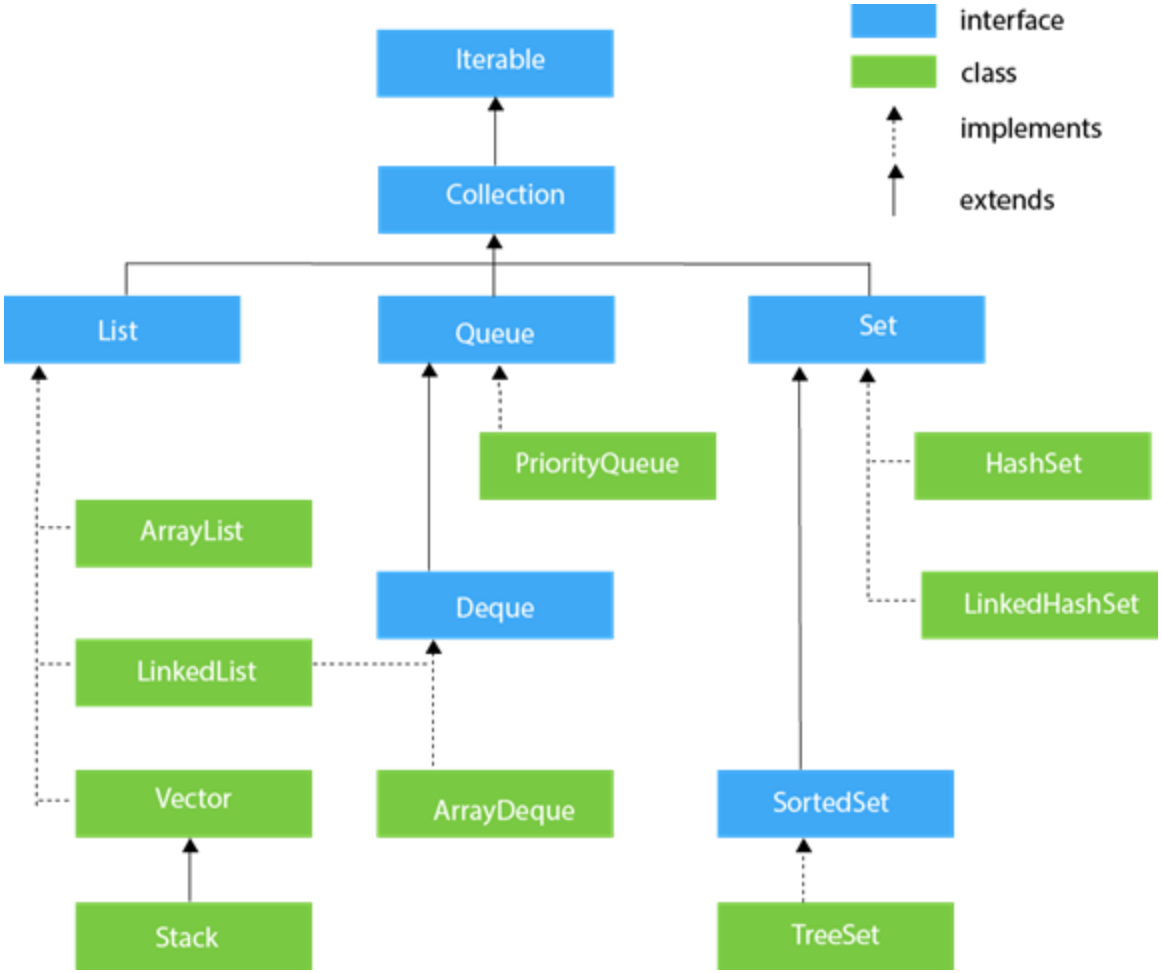
Like a basket for your data

Collections Framework? What's that?

- The **collections framework** contains interfaces and implementations of common data structures, like Vector, Set, List, and so on, which each represent a collection, or a container in which objects are stored
- The interfaces are split into two groups:
  1. Those which descend from **java.util.Collection**
  2. Those which descend from **java.util.Map**

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

Collection Framework Hierarchy



Collection vs. Map

- Interfaces which descend from Collection are **true collections**:
  - They have array representations
  - They can be operated on natively by iterative algorithms
- On the other hand interfaces which descend from Map are not true collections; instead they have collection-view operations which allow Collection-like manipulation
- Maps use key-value pairs, rather than indexes, and implementations are fundamentally different by design to implementations of Collections

The Iterable interface

- As the documentation says, implementing this interface allows the implementer to be the target of a for-each loop
- This is because Iterable demands three methods:
  - **iterator()**
  - **forEach()**
  - **spliterator()**
- With an implementation of iterator(), we can get an **iterator** that allows us to cycle through every element in an Iterable
- We can also use the enhanced for-loop, also known as the for-each loop:

```
for (Object o : objects) { /* do something */ }
```

The Iterator

- Every implementation of Collection has an **Iterator**, an object which facilitates iteration through the Collection:
  - Given a List list, you may call list.**iterator()**
  - Assuming you name the iterator it, you may call it.**hasNext()** to check if there is an additional element to iterate through
  - You may call it.**next()** to iterate to the next element
  - You may call it.**remove()** to remove the element the iterator is “pointing to”, or as the documentation says, the last element returned by next()

Collections and Generics

- Collections are only really useful if you also understand **generics**: the art of writing code for objects when you don't know or care what class the object is
- You'll know you're dealing with generics when you see these symbols: **< >**
- For example, this example of a list declaration:
  - ArrayList<String> al = new ArrayList<String>();
- String is the class that the ArrayList will contain
- When defining your own generics, you would use the type **T**, like so:
  - class MyContainer<T> { … }
- You would then use T as a class in the definition

Comparing Objects in Collections

- In the collections framework, we often find ourselves wishing to sort our collections. In order to do this, we have two options: implement Comparable in our objects, or create a Comparator
- The Comparable interface is used to compare an object to other objects of the same type. It indicates whether the other object is less than, equal to, or greater than itself
- Meanwhile, a Comparator compares two external objects of the same type. It can be passed as an argument for Collections.sort() to specify the sorting behavior

The Comparable<T> Interface example

```
public class Number implements Comparable<Number> {
        private int n;

        public int getN() {
                return n;
        }

    public int compareTo(Number other) {
                return this.n - other.n;
        }
}
```

Now we would be able to call the following on a List<Number>:

```
Collections.sort(list);
```

The Comparator<T> Interface example

```
public class NumberComparator implements Comparator<Number>{
        public int compare(Number n1, Number n2) {
                return n1 - n2;
        }
}
```

With this, we could now call the following on a List<Number>:

```
Collections.sort(list, new NumberComparator());
```

Discussion

- What is the difference between a collection, Collection, and Collections?
- What is a Map? Why is it different from a Collection?
- What is an Iterator? Would an iterator ever be useful?
- What's the difference between Comparable and Comparator?
- What are generics?
- How do you know you are dealing with a generic type?

Challenge

- **Goal:** Create a (simple) ArrayList class with the following characteristics:
  - It should have a private array of type Object
  - It should have a get method that gets the Object at the given index of the private array
  - It should have an add method that appends an Object to the end of the array
  - You must handle the case that your private array is not big enough to add
  - You must give the array some initial space

- **Stretch goal:** Make it generic!
  - Instead of Object, your (simple) ArrayList should use a generic type