Data Structures & Algorithms

Collection Framework

Amilcar Aponte
amilcar@cct.ie

Today's Plan

- Collections Framework

Collections Framework

- The Collections Framework is a grouping of Java code (interfaces & implementations) of data structures that hold collections of data
- Essentially, objects that can store other objects according to certain rules
- Things like Lists fall into this category

Extra reading:

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

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

Collection

- We could create a lot of these data structures ourselves (and you will next year) but for the moment, we want to look at how the existing ones are created and used.

- Collections allow us to do a number of things with the collection
  - Iterate through that collection (use a for each loop)
  - Split the collection up into definable chunks

Collection Interface

- boolean add(E e) - after this operation, one way or another, the element e is definately in there (we will see why so generic later)
- void clear() - removes all elements from the collection
- boolean contains(Object o) - returns true if the collection contains the object
- boolean containsAll(Collection<?> c) - returns true if the collection contains all the elements in the collection c

Collections Interface

- boolean isEmpty() - returns true if there are no more elements in the collection
- Iterator<E> iterator() - returns an iterator over the elements in the collection
- boolean remove(Object o) - removes a single instance of object o if it is in the collection
- boolean removeAll(Collection <?> c) - removes all this collection's elements that are also specified in the collection c

Collection Interface

- int size() - return the number of elements in the collection
- Object[] toArray() - returns an array containing all the elements in this collection
- <T> T[] toArray() - returns an array of type T (defined at runtime) containing all the elements in this collection

- Keep in mind these are not all of the methods. Only a few of them!

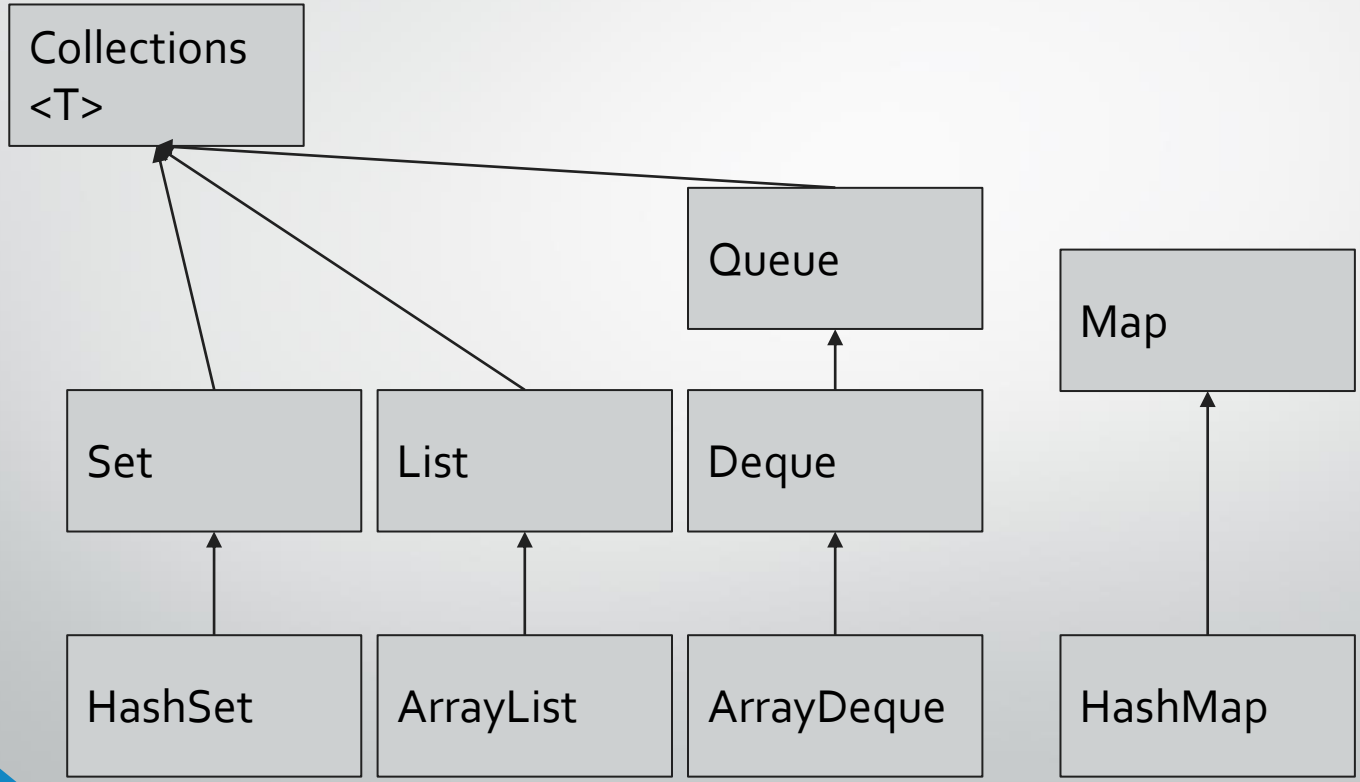
Types of Collections

There are a number of interfaces defined in the Collections Framework (other than the Collection Interface itself)

- Set
- List
- Deque
- Map

And each of these has some general purpose implementations defined

Let's see this graphically



Which one to use depends on your problem

- We use a set if we want a collection that does not allow duplicate entries (collection of employees or students perhaps)
- We use a list if we care about ordering and general access/manipulation
- We use a deque if we want a queue/stack-like structure, ordered contents with restrictions on addition and removal
- We use a map if we want to store key-value pairs

Set

- **A set is a collection that does not contain duplicate entries**
- Think of the mathematical definition of a set
- It only contains methods from the Collection interface itself and adds the no duplicate restriction

Some implementations:

HashSet, LinkedHashSet, TreeSet

List

- This is one that we've used many times
- A list is an ordered collection of data
- That ordering is specified by the user
- Elements can be accessed by their index (where they are in the list)
- Duplicate entries are allowed
- **There is no gaps in the collection**

Implementations:

ArrayList, LinkedList, Stack, Vector

Deque

- A Deque stands for a double-ended-queue
- This means that it is a linear structure that allows addition and removal from both ends of the structure
- We can add to front, or end, remove from front or end, or examine front or end
- There is no indexed access, unlike with the list

Implementations:

- ArrayDeque, LinkedList

Map

- Maps are not quite part of the Collections Framework but make use of some of the structures
- A map maps keys to values, so rather than be stuck with our array-like indexes to access data, we can use a pre-defined key
- For example rather than say Student[0] to get the name data stored in the first element we could say Student[name] to get the data
- We cannot have duplicate keys
- We need to be careful if we allow mutable elements

Implementations:

- HashMap, TreeMap, HashTable

That’s all folks

Any questions?