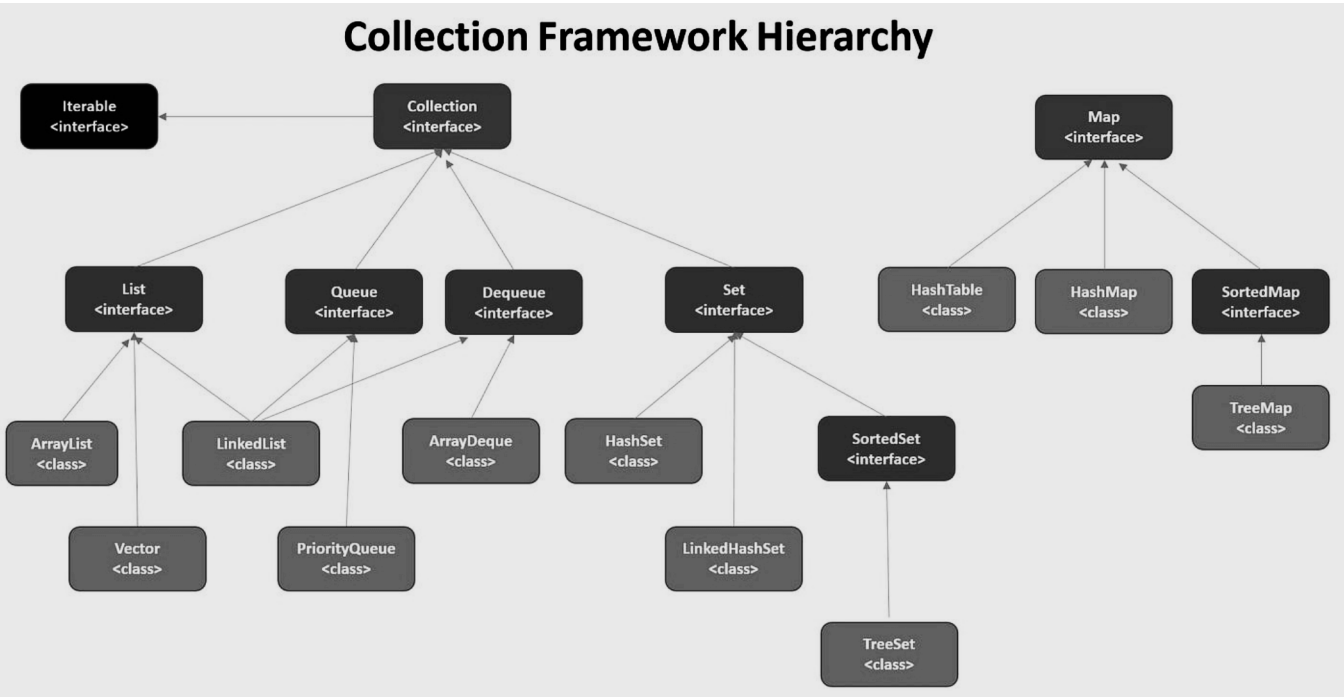

For more descriptive examples and stuff, look at Stroud's notes or the slides. Those were pretty helpful.

I. Stacks

Definition: stores data following the "First in, last out" philosophy. Things are placed at the top of the stack, and only the top is visible to the programmer.

Family: the java Stack class is built off of the List interfaces and considered a Collection; however, it is not really a stack because it allows for using indices to access its elements. Check the above hierarchy map.

- ⚠ Stroud considers the Stack class "deprecated" and encourages his students to use the Deque interface instead.
- ⚠ The programmer could also build a Stack from scratch using Nodes. Nodes are detailed below in the Linked Lists section.

Import: import java.util.Stack; import java.util.Deque; import java.util.LinkedList;

Nodes: Using a made-up Node class, the first node (top of the stack) will point to the next lower node, which will point to the next lower node, etc. Eventually, the last node (lowest on the stack) will point to null.

java.util.Stack methods:

- push(object) - place the object on top of the stack; returns the object that was just placed on the top
- pop() - take the top object off of the stack; returns the object the was just popped from the stack
- peek() - take a look but do not remove the top off the stack; returns the object on the top of the stack
- size(), toString() printed from bottom to top like an ArrayList using square brackets [], add(object) adds to back of the list (top of the stack), remove(index) and get(index)

java.util.Deque methods:

- addFirst(object) - place the object on top of the stack; returns the object that was just placed on the top
- removeFirst() - take the top object off of the stack; returns the object the was just popped from the stack
- peekFirst() - take a look but do not remove the top off the stack; returns the object on the top of the stack
- toString() prints from top to bottom like ArrayList using square brackets [], size()

II. Linked Lists

Definition: a group of nodes; each node contains a value and a reference to the next node in the list

Family: the java LinkedList class is considered a Collection, List, Queue and Deque. Check the above hierarchy map.

Classification:

- A singly linked list has each node point to the next node. The last element points to null.
- A circular singly linked list has the "last" node point to the first node, creating a circle.
- A doubly linked list is created by a "DoubleNode", which has a next and a previous node. Front of the list will have a previous pointing to null, while the back of the list will have a next pointing to null.
- A circular doubly linked list has the first and last nodes connected in a "doubly" manner, creating a circle.

Import: import java.util.LinkedList;

Methods:

- add(x), addFirst(x), addLast(x)
- add(x, y) - index x, value y
- get(x) - index x, getFirst() and getLast() - get the first and last elements of the list
- iterator(), listIterator()
- offer, offerFirst, offerLast
- peek, peekFirst, peekLast
- poll, pollFirst, pollLast - these are basically the same as the remove methods.
- contains()
- size()
- remove(), removeFirst() - remove front of list; remove(Object) - remove the first occurrence of Object, remove(index) - removes the element at this index
- clear() - remove all items from list
- toString() - prints something like [a, b, c, d], prints like ArrayList from left to right with square brackets []

Node: Node n = new Node(value, next);

- This is a made-up class by the programmer. Not from Java. But it models linked lists.
- This means that the Node n has a value of value, and points to the next Node. The last Node may point to null, or make a circle by pointing to an existing node.
- Has getter methods getNext() or next() and getValue() or get(), and setter methods setValue(x) and setNext(node), and may even have a toString() method that returns something like "a -> b -> c -> d"
- When traversing, do x = x.getNext() to move to the next node
- When removing nodes from a "list" of connected nodes, just set one node to skip a node and point to the next one. For example, [1, 2, 3, 4] turns into [1, 3, 4] if the 1 points to 3 instead of 2.

DoubleNode: DoubleNode k = new DoubleNode(value, previous, next)

- It has the Node methods, as well as the new methods getPrevious and setPrevious.
- When removing double nodes from a "list" of connected double nodes, just connect the unwanted node's previous and next nodes, before disconnecting the unwanted node from them.

III. Queues

Definition: a group of items all of the same type where items are added to the back of the queue and removed from the front. Works in a "first in, first out" manner.

- ⚠ Stroud considers the Queue interface deprecated and encourages his students to utilize Deque or write their own Queue classes
- ⚠ The slides claim that Queue only holds integers. This is outdated and misleading. Queue can hold any object.

Declaration: Queue<Integer> queue = new LinkedList<Integer>();

Import: import java.util.Queue;

Methods:

- add(x), where x is an object that is added to the back of the queue, sometimes called "enqueue"
- offer(x), basically same as add(x), with some insignificant differences
- remove() - removes the item at the front of the queue, sometimes called "dequeue"
- poll() - removes the item at the front of the queue, same as remove()
- peek() - returns the front item with no remove
- element() - same as peek(), but peek is used conventionally instead of element
- size() - returns # of items in the queue
- isEmpty() - tells whether queue is empty or not
- toString() - prints from front to end of queue, for example [1, 2, 3, 4] with square brackets [] (like arraylist)

Deque methods:

- addLast(object)
- removeFirst()
- The other LinkedList methods

IV. Priority Queues

Definition: queue structure that organizes the data inside by the natural ordering or by some specified criteria. The Java PriorityQueue is a min heap as it removes the smallest items (which have higher priority) first. It also stores Comparables.

Declaration: PriorityQueue<Integer> pq = new PriorityQueue<>();

Hierarchy: check out Stroud's notes for a conceptual understanding of PriorityQueue structure and function.

Import: import java.util.PriorityQueue; import java.util.Comparator;

Methods:

- add(x) or offer(x)- adds item x to the pqueue
- remove() or poll() - removes and returns the highest priority item (the smallest integer or ascii value)
- peek() - returns but does not remove the highest priority item (the smallest integer or ascii value)
- size() - returns # of items in the pqueue
- isEmpty() - tells whether pqueue is empty or not
- toString() - prints in a very interesting order which is demonstrated in Stroud's notes. Uses square brackets [] (like arraylist). [7, 11, 10] for example.

V. Hash Tables

Definition: maps keys to items, where hashCode() returns a hash code value for each object. The hashcode may be the same as the value of an item or may involve the use of some elaborate formula in hopes of creating a unique key. It is similar to HashMap but slower; it is now considered obsolete.

hashCode(): In java, every Object has a hashCode() method that returns a hash code value for that Object.

- For Characters, it returns the ascii value
- For Integers, it returns the same integer
- The programmer may implement the hashCode() method and change its insides in a user defined object class
- In most hash tables, the key involves some manipulation of the hash code and the value is just the thing being stored.
    - ➢ In the examples, this is typically done by doing object.hashCode() % n, and using this for the index in an array of size n that represents the hash table.

LinkedList: most hash tables are built using an array of linked lists.

- For example, LinkedList[] table = new LinkedList[10];
- It is also possible to have an array of user-defined Node objects (which model LinkedLists) for a hash table

Collisions: this happens when you get the same index position for several values.

- ☆ Solution: simply store the values with the same hash code into a list at that index position.
- ☆ The bigger the table (like a LinkedList array with size 1000), the less likely it is to have collisions.