Use the following outline to guide your self-assessment and notetaking

Week 9 – Collections

Collections and Data Structures (Ch 13.1)

Separate interface from implementation

- Collections can be implemented with various techniques
    - Arraylists are implemented based on arrays
    - LinkedLists uses a special kind of data structure based on its appropriate name (linked list)
    - Abstract data type: Collection of data and operations allowed on that data
        - Contains a name, domain of values, and set of operations
        - Operations are separate from the implementation
    - Objects have well defined interfaces hidden in the class, which allows for reusability and reliability because the interaction with the rest of the system is controlled (think data encapsulation)

Dynamic Representations (Ch 13.2)

Dynamic Structures

- These do not have predefined sizes and can easily grow and shrink as data changes
- Arraylists are not efficient because the implementation works by creating a larger array and then copying everything over when necessary

Dynamically linked list

- Manipulates object references to link data together
    - A node object could contain a reference to another node object, which is the key to creating a linked list
    - Last node would have a null reference
    - Insert and delete operations are implemented by manipulating the object references

Other dynamic list representations

- Linked lists can also have a reference to the previous node (a doubly linked list)
- A header can be used to have a reference to the front of the list and another reference to the rear of the list
    - Useful for manipulating data in the front or in the rear
    - Can store other information like the number of nodes
- Use of a header can even be combined with a doubly linked list

Linear Collections (Ch 13.3)

Queues

- Similar to a linked list except it has restrictions on the way you insert and delete elements

    - FIFO: First in First out order
    - Items that you put in will appear at the end of the queue
    - You take items out from the beginning of the queue (first item you put in, first item comes out)
        - Kind of like people waiting in line, hence the name (queue)
        - Customer enters at the back and moves forward as earlier customers are serviced
    - Operations:
        - Enqueue: Add item to the back of the queue
        - Dequeue: Pops an item from the front of the queue
        - Empty: Return true if the queue is empty or false otherwise

Stacks

- This time, elements enter and return from the same end
    - LIFO (Last in first out)
- Last item entering the stack is the first item to come off of the stack as well
    - Think of stacking bricks
    - The top brick is the first one to be removed from the stack
- Operations:
    - Push: Push an item on top of the stack
    - Pop: Remove the item from the top of the stack
    - Peek: Retrieves the item on top of the stack without removing it
    - Empty: Return true if the stack is empty or false otherwise