Summery for Capsule 2: Lists and Iterators
LinkedList:

- Data structure like Array or ArrayList
- Comprised of nodes
    - Each node has a value
    - Each node has a pointer reference to another node, to advance down the list
        - The last node does not have a pointer normally
        - The last node could point to the first node to create a circularly LinkedList
    - Each node could have a second pointer, pointing back to the previous node
        - This style of LinkedList is called a Doubly LinkedList
- ListIterator
    - Java's own given interface
    - An iterator describes a position anywhere inside a LinkedList
    - Has a .next(), which moves down the LinkedList
    - Has a .hasNext() to check if there is another node in the LinkedList
    - Has a .hasPervious() to check if there is a node before the node which the iterator is at (useful for doubly LinkedLists)
    - Has a .add() which adds a new node after the node which the iterator is at
    - Has a .remove() which removes the node and returns its value
- Traversing a LinkedList and accessing an element
    - To get any one element of a LinkedList takes O(n) iterations (random access)
        - Need the use of an iterator to keep track of where you are in the LinkedList
    - Whereas to get to any one element in an Array takes only O(1) iterations
- Adding or removing an element
    - In ArrayLists it takes O(n) iterations to remove or add an element
    - In LinkedList it takes O(1) iterations to remove or add an element
    - Removal of a node
        - Use an iterator to find the node which is before the node to be removed
        - Take that node and change its pointer to point to the node which is after the node to be removed
        - Java should then garbage collect the node which has no pointers pointing to it
        - Thus removing the node from the LinkedList
        - Could also use the iterator's .remove() method
        - If removing the first or last node
            - Can use LinkedList's .removeFirst() or .removeLast() methods
    - Adding a new node
        - Use an iterator to find the node which is before the location of the new node
        - Create a new node
            - Give this new node's pointer a value by making it point to the node after the one the iterator is on
        - With the node the iterator is on, have its pointer value point to the new node

- Thus adding a new node to the LinkedList
- Could also use the iterator’s .add() method
- If adding something to the beginning or end of the List
    - Can use LinkedList’s .addFirst() or .addLast() methods