Capsule Project: Lists and Iterators

Iteration is the traversing of and interaction with the objects stored in a given container. There are several types of containers in the Java API (such as a LinkedList or ArrayList) and several methods of iterating through each.

In most container classes in the Java API, each object is stored with its index such that if a user wishes to search through the List, they can simply create a for or while loop to examine the object at each index in the List. Alternatively, there exists an Iterator interface that, when implemented and instantiated in a container class (as it already is in most Lists in the Java API), can traverse and interact with the contents of that List.

When used as-is directly in a List, an Iterator has three methods (hasNext(), next(), and remove()) and will sit between two indices and iterate (forward only) through each object in the List as the code tells it to, returning each object it passes on the way or removing the object it just passed.

Additionally, there is a ListIterator interface that implements the vanilla Iterator and adds many extra methods. When it is told to, a ListIterator can move forward or back, return the object that is in front of it or behind it, add new objects, remove old objects, or replace existing objects with new ones.

ListIterators are often used to traverse LinkedLists, which are very similar to ArrayLists in function and use but have several extra methods that make them especially useful as stacks or queues.

```
Instantiating and initializing a LinkedList:
  LinkedList<String> inventory = new LinkedList<String>();
  inventory.add(“Doll”);                              //adds the string “Doll” to the LinkedList
  inventory.add(“Yo-yo”);                                 //appends the string “Yo-yo” to the end
  inventory.addFirst(“Jacks”);                     //inserts the string “Jacks” at the beginning
  inventory.add(2, “Car”);                        //inserts the string “Car” at the second index
  inventory.toString();                     //returns a string reading “Jacks, Doll, Car, Yo-yo”
Other LinkedList methods: get(int index), getFirst/Last(), indexOf(Object o)

Creating a ListIterator and traversing a LinkedList:
  ListIterator<String> inventoryIterator = inventory.listIterator(0);
                   //Creates a new ListIterator in front of the String at inventory’s 0th index
  inventoryIterator.hasNext();                                                    //returns true
  inventoryIterator.hasPrevious();                                               //returns false
  inventoryIterator.next();
              //returns “Jacks” and moves the ListIterator to the next index in the LinkedList
  inventoryIterator.remove();       //removes the previous object (“Jacks”) from the LinkedList
  inventoryIterator.add(“Transformer”);
     //inserts the string “Transformer” after the last returned object (the start in this case)
  inventoryIterator.next();                            //returns “Doll” and moves the ListIterator
  inventoryIterator.set(“Nerf gun”);
                            //replaces the previous object (“Doll” here) with the given object
  inventory.toString();          //returns a string reading “Transformer, Nerf gun, Car, Yo-yo”
Other ListIterator methods: nextIndex(), previous(), previousIndex()

Shorthand notation of for loops when iterating through lists:
  for (ObjectType variable : nameOfList) {
     //Traverses the list, setting the variable to the current object each time it loops
  }
```