**Iterator Worksheet**

Preparation for IteratorLab

The Collection Framework requires that some Collections, including Lists and Sets, contain an inner, private object, called an "**iterator**," that allows the data structure to traverse through its data items one-by-one. Programmers do not need to know how the data is actually structured. All the programmer needs to know is how to call the iterator’s methods, which is what the API is for. That’s the point. This lab introduces you to iterators and what they can do. In fact, you have used iterators before when you used the Scanner class to read data from a file.

1. Assume that we have instantiated a concrete subclass of List<E> or Set<E> named theList which stores Student objects. Write the code to instantiate an iterator:
2. Now use the iterator’s methods to println every item in that concrete class:
3. The it.remove() method *“removes the last element that was returned by next.”*
4. Look for ListIterator<E> on the cheat sheet. You can instantiate a ListIterator<E> from either ArrayList<E> or LinkedList<E>. It provides two more methods, add(E) and set(E).
5. Read the cheat sheet comments on the ListIterator methods. What does this language mean? *it.add(E) "adds obj before the element that will be returned by next."* They are explaining how iterators behave by imagining a pointer which points \*between\* two data items.
6. Given: theList ["A", "B", "C", "D", "E"]  
   Given: ListIterator<String> it = theList.listIterator();

Draw the imaginary pointer before the first item in list theList: ^**A B C D E**

Given these commands, show the list and the pointer:

it.next(); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

it.next(); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

it.remove();//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

it.next(); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

it.add("X"); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Let lsit be a ListIterator<E> on myList that contains two Strings ["A", "B"].

After each command, show the state of the iterator lsit:

ListIterator<String> lsit = myList.listIterator();//\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lsit.add( X ); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lsit.next(); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lsit.add( Y ); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lsit.next(); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lsit.add( Z ); //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lsit.next(); //NoSuchElementException

1. A ListIterator<E> has an add(E) method that *"inserts the element before the element that will be returned by next."*
   1. If a ListIterator is empty, does add(E)still work? Y/N
   2. If a ListIterator has the pointer before the first element, does add(E) still work? Y/N
   3. If a ListIterator has the pointer after the last element, does add(E) still work? Y/N

1. Assume al is an ArrayList<String> and you have to write code to delete every "Bob"   
   from al. Below are three solutions. Decide if each works or not. If it doesn’t work, name the Exception it throws.

Student A:

for(int i = 0; i<al.size(); i++)

if( al.get(i).equals("Bob") )

al.remove( i );

Student B:

ListIterator<String> li = al.listIterator();

while( li.hasNext() )

if( li.next().equals("Bob") )

li.remove();

Student C:

int position = 0;

for(String str: al)

{

if(str.equals("Bob"))

al.remove( position );

position++;

}

1. A *for-each* loop or an Iterator: how do you choose?
2. Using an Iterator or a ListIterator: how do you choose?
3. Code written before Java 1.5 does not change (wrap) int to Integer automatically. The programmer had to do that every time. Code written after Java 1.5 wraps int to Integer automatically. Since we are currently using Java 1.8, you don't have to worry about that.

**IteratorLab Assignment**

Complete the IteratorLab shell, using either for-each loops or iterators, not regular for-loops. Notice Lines 11-12, where they made the for-each loop backwards compatible with primitives. The replaceNeg method on Line 17 replaces all negative numbers with 0.