# Exercise 1

By now, you should have watched the videos on the course website on interfaces Iterator and Iterable. To refresh your memory, let’s do a warm-up exercise.

Consider the class below:

public class ILinkedList<E> implements Iterator<E>{

private LinkedList<E> list;

private int n;

public ILinkedList(E[] input) {

list = new LinkedList<E>();

for (E e: input) {

list.add(e);

n++;

}

}

@Override

public boolean hasNext() {

// TODO Auto-generated method stub

return n > 0;

}

@Override

public E next() {

if (!hasNext()) throw new NoSuchElementException("done");

n--;

E next = list.get(n);

return next;

}

public static void main(String[] args) {

String[] l = new String[]{ "string1", "string2”, "string3", "string4", "string5"};

ILinkedList<String> myList = new ILinkedList<String>(l);

while (myList.hasNext()) {

System.out.println(myList.next());

}

}

Write the output to the console of running this application in Eclipse below:

**SOLUTION**

**string5**

**string4**

**string3**

**string2**

**string1**

# Exercise 2

Now, you will implement Iterator yourself. Consider the beginnings of a class TreeList below:

package demo;

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.NoSuchElementException;

/\* An instance of TreeList has a list of elements and supports iteration

\* over the elements of its list in the following fashion. If this TreeList

\* has a list a[0..n], the elements are returned by the iteration in the

\* order a[0]-a[n-1]-a[1]-a[n-2]-... until each element is returned exactly

\* once.

\*/

public class TreeList<E> implements Iterator<E> {

private List<E> list; // contains the elements of this TreeList

private int n; // contains the index of the next 'list' element in the iteration

private boolean firstHalf; // true if n points to the first half of 'list'

//(includes the middle element when list is odd-sized.

public TreeList(E[] input) {

list = Arrays.asList(input);

n = 0;

firstHalf = true;

}

/\*\* TODO implement

\* Returns true if the iteration has more elements.

\* (In other words, returns true if next would return an element

\* rather than throwing an exception.)

\*/

@Override

public boolean hasNext() {

if (list.size()%2 == 0)

return firstHalf? n != list.size()/2 : true;

else

return !firstHalf? n != list.size()/2 : true;

}

/\*\* TODO implement

\*@spec Returns the next element in the iteration based on the formula

\* described in the class comment above.

\*@return the next element in the iteration

\*@throws NoSuchElementException - if the iteration has no more elements

\*/

@Override

public E next() {

if (!hasNext()) throw new NoSuchElementException("done");

E next = list.get(n);

if (firstHalf)

n = list.size() - n - 1;

else

n = list.size() - n;

firstHalf = !firstHalf;

return next;

}

public static void main(String[] args) {

String[] l = new String[]{"string1", "string2", "string3", "string4", "string5"};

TreeList<String> myList = new TreeList<String>(l);

System.out.println("Even sized list first");

while (myList.hasNext()) {

System.out.println(myList.next());

}

System.out.println("Odd sized list second");

l = new String[]{"string1", "string2", "string3", "string4", "string5", "string6"};

myList = new TreeList<String>(l);

while (myList.hasNext()) {

System.out.println(myList.next());

}

}

}

/\*

\* (a) First,let's make sure you understand the iteration order.

\* Assume hasNext() and next() have already been implemented

\* according to the specifications. Write below the output of

\* running this application (method main) in Eclipse below.

\* - Even sized list firt

\*

\*

\*

\*

\*

\*

\* - Odd sized list second

\*

\*

\*

\*

\*

\* (b) Now, implement hasNext() and next() according to the specification.

\*/

# Exercise 3

We now want to be able to write a for-each loop on our TreeList. To do so, it needs to implement Iterable. We will then split our TreeList into two new classes: ITreeList and TreeIterator. ITreeList will implement Iterable so that we can write a for-each loop for it. TreeIterator will be a private class within ITreeList and will provide the hasNext and next methods by implementing Iterator. Class stubs are provided below, with clas constructions implemented for you as a starting point. Implement the other methods.

/\* An instance of TreeList has a list of elements and supports iteration

\* over the elements of its list in the following fashion. If this TreeList

\* has a list a[0..n], the elements are returned by the iteration in the

\* order a[0]-a[n-1]-a[1]-a[n-2]-... until each element is returned exactly

\* once.

\*/

public class ITreeList<E> implements Iterable<E> {

private List<E> list; // contains the elements of this TreeList

public ITreeList(E[] input) {

list = Arrays.asList(input);

}

@Override

public Iterator<E> iterator() {

return new TreeIterator();

}

public static void main(String[] args) {

String[] l = new String[]{"string1", "string2", "string3", "string4", "string5"};

ITreeList<String> myList = new ITreeList<String>(l);

System.out.println("Even sized list first");

for (String element: myList) {

System.out.println(element);

}

System.out.println("Odd sized list second");

l = new String[]{"string1", "string2", "string3", "string4", "string5", "string6"};

myList = new ITreeList<String>(l);

for (String element: myList) {

System.out.println(element);

}

}

private class TreeIterator implements Iterator<E> {

private int n; // contains the index of the next 'list' element in the iteration

private boolean firstHalf; // true if n points to the first half of 'list'

//(includes the middle element when list is odd-sized.

public TreeIterator() {

n = 0;

firstHalf = true;

}

@Override

public boolean hasNext() {

// same as in exercise 2

}

@Override

public E next() {

// same as in exercise 2.

}

}

}