# Laboratory 9: Iterators, Ordered Lists, and Searching

## Part A: Iterators

An iterator is an object that allows you to access the items stored in a data structure sequentially. This has two major advantages. The first advantage is that because many very different kinds of data structures have iterators defined for them, code can be written that will work independent of the choice of data structure. That code is protected against changes in data structure. For example, using Java's ListIterator, here is code that will remove all items from a data structure holding items of type X.

```
ListIterator<X> toClear = someDataStructure.getIterator();
while(toClear.hasNext())
{
      toClear.next();
      toClear.remove();
}
```

As long as the object someDataStructure has implemented the method getIterator(), the rest of the code is insulated from change. Since sequential access to a collection of items is very common, iterators are fairly useful. The second advantage is that the iterator may be specialized to provide fast sequential access to the items in the collection. For example, consider a list that uses a linked chain. The items in a simple singly-linked chain could be accessed one at a time using a get-entry method. The only problem with this is that each time a get-entry executes, the chain must be traversed from the front. An iterator would be able to keep a reference to the nodes in the linked chain and would not have to restart from the beginning for each access.

The following steps should be completed in the lab9Driver program or in the class files it uses.

**0) Demo an Iterator for Vector**

**1) Add an iterator capability to AList**

**2) CharIterator -- An Iterator that works on String**

A fun way to get started with Iterators it to make your own Iterator class that iterates through the letters of the String provided to its constructor. Open CharIterator and write and test this Iterator class that works on Strings. The constructor will receive the String to be iterated on. The next method will return the next Character from the String. The method hasNext will indicate if any letters remain. The method remove will remove the letter just returned by next from the String (it will also have to modify cursor so it does not skip over the next letter). Each value returned should be an object of type Character.

**3) Solitaire – A matching game using Iterator**

Consider a solitaire matching game in which you have a list of random integer values between 10 and 99. You remove from the list any pair of consecutive integers whose first or second digits match. If all values are removed, then you win.

For example, consider the following sequence of 10 integers:
10 82 43 23 89 12 43 84 23 32

The integers in the pair 10 and 82 do not match in either digit and so cannot be removed. However, the integers in the pair 43 and 23 match in the second digit and are removed, leaving the following sequence:

10 82 89 12 43 84 23 32

Continue checking for pairs from 89, the value after the removed pair. No other pairs have matching integers. Now return to the beginning of the list and check the pairs. The integers in the pair 82 and 89 match in the first digit and can be removed:

10 12 43 84 23 32

And now that 82 and 89 are gone, we find that 10 and 12, which are now adjacent, can also be removed:

43 84 23 32

No other pairs can be removed, so we lose.

Open Solitaire and write a program that simulates this game. It should generate 20 random two-digit integers and place them in an instance of **java.util.ArrayList**, using an instance of **ListIterator**. Then, using this iterator, scan the list and remove matching pairs of values. After each pair is removed, use an iterator to display the values remaining in the list. You may wish to read the Javadocs for ArrayList and ListIterator which you can google with "java ArrayList" etc.

Here is an example “winning” list (you should be able to delete all of the numbers using your algorithm):

34 10 82 89 12 84 23 53

4) Extend a SortedAList class from the AList class using inheritance. Define add, getPosition and remove and verify these work and maintain the list items in sorted order.

5) Solve the PacketReader program described in part B below.

## Part B: Sorted Lists

**PacketReader**

In certain computer networks, a message is not sent as a continuous stream of data. Instead, it is divided into pieces, called packets, and sent a packet at a time. The packets might not arrive at their destination in the same order as the one in which they were sent. To enable the receiver to assemble the packets in their correct order, each packet contains a sequence number.
For example, to send the message “Meet me at 6 o’clock” three characters at a time, the packets would appear as follows:

1 Mee
2 t m
3 e a
4 t 6
5 o'
6 clo
7 ck

Regardless of when the packets arrive, the receiver can order the packets by their sequence numbers to determine the message.

Open the file message.txt in the lab9 project folder. Given this text file containing the packets of data in the order they were received, write an application that reads the file and extracts the message by using a sorted list.

Write your application in PacketReader. Below the PacketReader class implement the auxiliary class Packet, which represents a single packet as a combination of a String for the data and an int for the sequence number. Define a read method similar to the SlideShow read from lab9 that reads a packet from a file. In order to make a SortedList of Packet you'll have to define a compareTo method and indicate the class implements Comaparable<Packet>.

Write statements to read the packets from message.txt and insert them into a SortedArrayList. Use the getEntry method for SortedArray to print the packets after they have been assembled. The output should not show any packet breaks.

## Part C. Search

### 6) SearchRange

Consider an array data of *n* numerical values in sorted order and a list of numerical target values. Your goal is to compute the smallest range of array indices that contains all of the target

values. If a target value is smaller than data[0], the range should start with -1. If a target value is larger than data[n - 1], the range should end with *n*.

For example, given the array {5 8 10 13 15 20 22 26} and the target values (8, 2, 9, 17), the range is -1 to 5.

**a.** Devise an efficient algorithm that solves this problem. Hint: there is a binary search method in the Java Collections class. Click here for more information.
**b.** If you have *n* data values in the array and *m* target values in the list, what is the Big Oh performance of your algorithm?
**c.** Implement and test your algorithm.

## Part D. LinkedList Iterator

7) Modify LinkedList so it has iterator capability as well. Test it using the test routine in step 1