CSC 102 Lecture Notes Week 7
More on Data Structures and Abstract Data Types

I. **Relevant reading.**

A. Chapter 10 (again, for Program 5)

B. Chapter 15

II. **Midterm and Lab Quiz this Wednesday**

A. Covers topics on sample plus exceptions.

B. No file i/o.

C. Possibly Comparable interface.

D. No searching or sorting.

III. **Near-Term Labs and Program**

A. Lab 13 -- *Implement array list*

B. Lab 14 -- *explore linked list*

C. Program 5 -- *maze file i/o*

D. See the writeups for details

IV. **The Java Library Collection Hierarchy**

A. Figure 1 is a UML diagram of key classes in the Java *collection hierarchy.*

1. It's a high-level diagram focusing two specific interfaces and two specific classes
2. It leaves a good deal of detail of intermediate interfaces and abstract classes not intermediately relevant to the discussion at hand.
3. Note how the diagram shows that an interface can extend another interface, as in the *List* interface

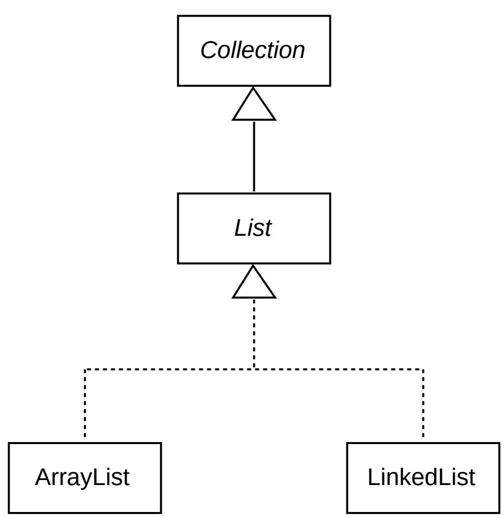

Figure 1: Four key components of the Java collection hierarchy.

extending the *Collection* interface.

B. During class we'll walk through the Java library documentation for this class/interface hierarchy.

1. We'll start at with the ***Collection*** interface the top.
2. Then we'll look the ***List*** interface, which adds methods to impose ordering on a collection.
3. We'll then walk through the documentation for two particular classes that implement the ***List*** interface -- ***ArrayList*** and ***LinkedList***

C. In upcoming labs and programs you'll be implementing your own simpler versions of Java library lists.

V. **Introduction to Data Structures (Ch 15)**

A. A primary purpose of a data structure is to support ***efficient computation*** in a computer program.

1. Efficiency is measured using the "Big-Oh" notation introduced in Notes 6
2. In Notes 7, we'll discuss further some efficiency *trade offs* for different types of data structure, in particular ArrayList and LinkedList.
3. A *trade-off* means that different data structures are efficient for certain operations, but inefficient others.
   a. However, no single structure may be efficient for all operations.
   b. Hence, when choosing a data structure to use for a particular program, one must consider carefully the problem to be solved, and choose the data structure the provides the best efficiency available for that problem.
4. The table in Notes 6 illustrates comparative efficiencies for array versus linked structures.

B. **A Brief Review of arrays and ArrayLists.**

C. You've used Array structures a good deal in CSC 101 and so far in 102.

D. C provides plain arrays, Java provides both arrays and the more flexible ArrayList class.

E. At its core, an ArrayList uses an array to store data.

VI. **The Basic Idea of a `LinkedList` data structure (Sections 15.1, 15.2 of the book).**

A. In an `ArrayList`, elements are stored in a single sequential block.

B. In a `LinkedList`, elements are in separate ***nodes***, with links referring to neighboring elements.

C. Figure 2 is a picture of what array and linked structures look like, in particular for the simple program ArrayListAndLinkedListExample.java from Notes 6.

D. In class, we'll walk through the library API for `java.util.LinkedList`

VII. **Iterators, Particularly for Lists (Section 15.1 of the book)**

A. The purpose of an *iterator* in Java is to provide the means to return all the elements of a collection, without having to know the underlying data structure within the collection.

B. The examples in Chapter 15 of the book provide a good introduction.

C. Java provides an *Iterator* interface with these key methods:

1. ***next*** -- return the next element in a collection
2. ***hasNext*** -- return true if not at the end of a collection

D. There is also a *ListIterator* interface with these key methods:

1. ***next*** -- return the next element in the list
2. ***hasnext*** -- return true if not at end of the list
3. ***previous*** -- return the previous element in the list
4. ***hasPrevious*** -- return true if not at the start of the list

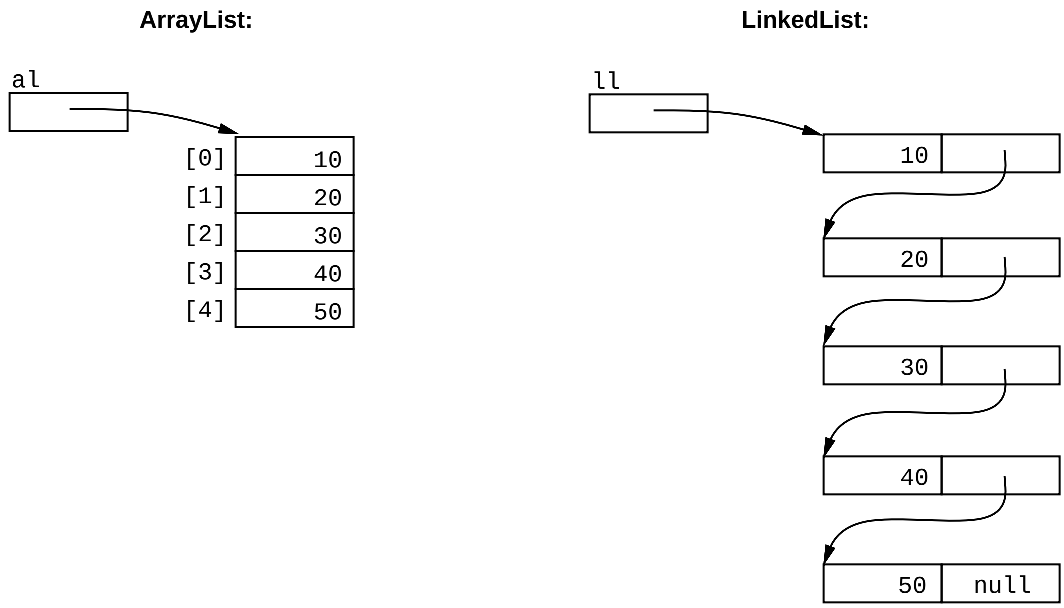

Figure 2: Array and linked list data structures.

E. All Java *List*s implement the ***listIterator***, meaning they must provide all of its methods.

1. A list iterator starts at beginning of list and returns elements in sequential order.
2. Iterators can be used conveniently in a for loop, as in the following example that prints all the persons in a person list:

```
for (Person p : PersonList) {
    System.out.println(p);
}
```

F. There are further code examples from the book in 102/examples/book/ch15/impllist/ListIterator.java and 102/examples/book/ch15/impllist/ListTester.java

VIII. **Two More Widely-Used Data Structure -- Stacks and Queues (Section 15.4 of the book).**

A. Stacks and queues are used all over the place in programming.

B. Section 15.4 of the book is a good introduction.

C. A stack is a last-in-first-out -- ***LIFO*** data structure.

D. A queue is a first-in-first-out -- ***FIFO*** data structure.

E. The key stack operations are:

1. ***push*** -- add to the top
2. ***pop*** -- remove from the top
3. ***peek*** -- get top, no remove

F. The key queue operations are:

1. ***enqueue*** -- add to the end
2. ***dequeue*** -- remove from the front
3. ***getFirst*** -- get first, no remove

IX. **List Implementation of Stacks and Queues**

A. Should we use and ArrayList or LinkedList to implement stacks and queues?

B. In class, we’ll consider the operations of each and examine the implementation trade offs.

C. You’ll be covering these two data structures in much more detail in CSC 103.