Section 8:
Linked Lists

Pair up with anyone who is agreeable to pairing up with you, pick the first driver, and let’s get to work!

PART I: A Generic Node Class

You know what a node contains: A data field (usually a reference to an data object) and a next field (a reference to the next node in the list).

1. On the class web page is a file named Section8.java. Load it into Dr. Java.

2. At the top is a class named AnimalNode. That’s the class that we will use to create data objects for our linked list’s nodes to reference. Below that is a start on the Sec8Node class, which, as you can see, uses generics so that a Sec8Node, and a list built of Sec8Nodes, can reference any kind of object, not just AnimalNode objects.

Your main task for this checkpoint is to complete Sec8Node. It needs the usual items for a node class:

- A data instance variable and a next instance variable.
- A constructor that accepts one formal parameter, a reference to the generic data object, and sets the data instance variable to it, and sets the next reference to null.
- Two getters, getData() and getNext().
- Two setters, setData() and setNext().

That’s it; code the Sec8Node class!

3. Compile and run Section8.java. If you see any assertion exceptions before the Testing Sec8List... line, you have some debugging to do. (And, remember that you have to enable assertions when you run this code, like you did for Section6.java.)

CHECKPOINT 1
Raise your hand. Your SL will come over and verify that you are passing the Sec8Node tests.

PART II: Tracing Linked List Code

Debugging code by hand is an important skill to develop. Often, it’s the best way to really understand how your code is behaving. In this checkpoint, you and your partner will trace the execution of two methods that work with linked lists.

1. Section8.java contains the Sec8List<E> class, which is (no surprise) a linked list class. Take a few minutes to examine the instance variables and methods that Sec8List<E> already possesses.

(Continued ...
2. Consider this potential `Sec8List<E>` instance method:

```java
public void unknown () {
    Sec8Node<E> newHead = null, temp = head, newtemp;
    while (temp != null) {
        newtemp = new Sec8Node<E>(temp.getData());
        newtemp.setNext(newHead);
        newHead = newtemp;
        temp = temp.getNext();
    }
    head = newHead;
}
```

Your task: On paper, trace the execution of this method. You need to learn two things about it:

(a) What is the task that it is supposed to accomplish?
(b) Does it do that task correctly on lists of 0, 1, 2, and 3 nodes?

Be sure to write down your answers; your SL will want to know.

3. Here’s another method:

```java
public void frontToRear () {
    Sec8Node<E> newHead, fore = head, aft = null;
    while (fore != null) {
        aft = fore; fore = fore.getNext();
    }
    newHead = head.getNext();
    aft.setNext(head);
    head.setNext(null);
}
```

`frontToRear()` is supposed to detach the first node of the list and re-attach it as the last node of the list. It doesn’t work. Your first task for this method: Trace it to learn why it doesn’t work.

4. When you think you see this method’s problem(s), write down what you think needs to be added or changed so that it performs that task correctly, no matter how many nodes are in the list.

5. Let’s find out if your fixes work! Type the `frontToRear()` method into `Sec8List<E>` in the location indicted by the comments, and compile and run `Section8.java`. There’s a section in the output labeled “Testing `frontToRear()`...”. Your goal is to pass the tests in that section.

[✔] CHECKPOINT 2 Raise your hand. Your SL will come over to check your answers for the first tracing and check the correctness of your `frontToRear()` method.

(Continued …)
PART III: Coding Linked List Methods

Following linked list code is a good skill, but writing it is the next step toward mastering linked lists.

1. In Sec8List<E> is a method stub for toArrayList(). Its job is to construct an ArrayList of the data objects held by the Sec8List<E> list and return it. The data objects should be in the same order within the ArrayList as they are in the Sec8List<E> list. Write toArrayList().

2. Compile and run Section8.java. If your method doesn’t pass the toArrayList() tests, find and fix your error(s) before continuing.

3. Sec8List<E> also has a stub for makeShallowCopy(), whose job is to create a new Sec8List<E> list that has the same data objects as the original list. That is, your method should do something similar to the last thing your section leader talked about with the concatenate operations at the start of section today. This type of copying is called a shallow copy, because we are only copying the data structure, not the data stored in it. Write makeShallowCopy().

HELPFUL HINT: You may find your toArrayList() method, and the supplied prepend() method, to be useful.

4. Compile and run Section8.java. If your method doesn’t pass the makeShallowCopy() tests, find and fix your error(s) before continuing.

✔ CHECKPOINT 3 Raise your hand. Your SL will come over and verify that your methods pass the tests.

PART IV: Clean Up!

1. Log out of your computer.

2. Pick up your papers, writing implements, cell phones, trash, etc.

3. Push in your chair(s).

✔ CHECKPOINT 4 Raise your hand. Your SL will come over and

You’re free to go! But, if you have time, we recommend that you use it to work on the next programming assignment.