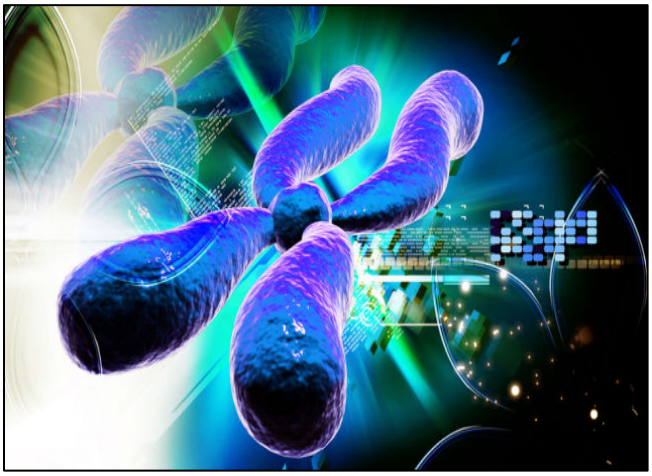
CS 46B
Fall 2019
Lab 88: Transposons



In this lab you will model the activity of transposons. Transposons are short regions of chromosomes (DNA molecules) that are believed to play a role in the development of certain kinds of cancer, including lung and throat tumors.

As always: Work in pairs. One of you will be the "driver", who types into Eclipse. The other is the "scribe". Alternate jobs, week by week.

You will both submit lab reports.

Driver: At the top of your lab report write

CS 46B Lab Report
Your name
I was the driver, Xxxx Yyyy was the scribe.

Scribe: At the top of your lab report write

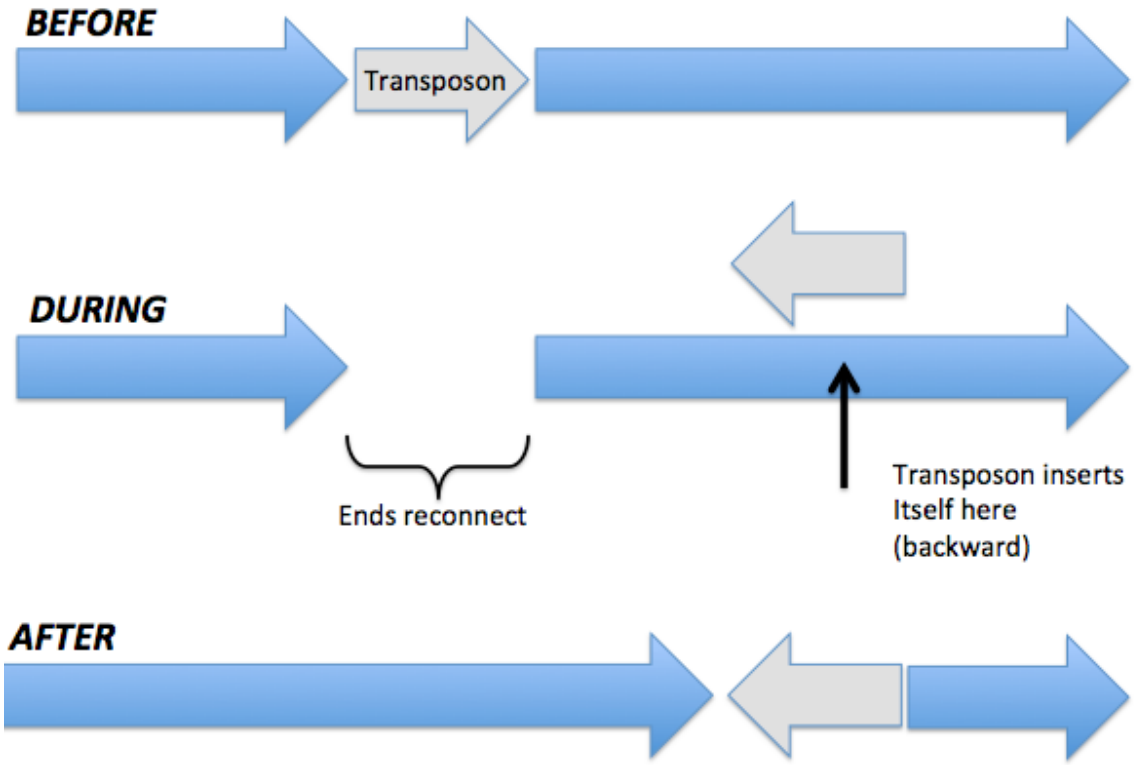
CS 46B Lab Report
Your name
I was the scribe, Xxxx Yyyy was the driver.

As you work through the lab assignment, instructions/questions highlighted in yellow should be discussed by both of you and addressed by the driver in the driver's report; instructions/questions highlighted in blue should be discussed by both of you and addressed by the scribe in the scribe's report.

Background

Recall that a chromosome is a long molecule of DNA. DNA is a long chain composed of the chemical subunits (or "bases") adenine, cytosine, guanine, and thymine, which are usually abbreviated as A, C, G, and T. In software, chromosomes are often modeled as strings of these characters. Since chromosomes can be hundreds of millions of bases long, the strings that model chromosomes can be hundreds of millions of characters long.

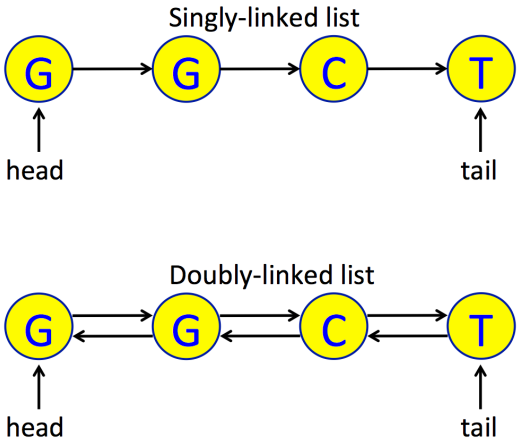
Sometimes a relatively short segment of DNA, called a transposon, drifts out of its chromosome for no apparent reason, turns itself around, and inserts itself backward into another part of the chromosome. The space left at the segment's original location closes up again.



Often transposons do no harm. Sometimes they kill their cell, by inserting themselves in the middle of a unit of DNA (a gene) that programs some vital cell function. The function gets disrupted and the cell dies. But the death of 1 cell is no problem; you and I have around 60,000,000,000,000 cells. However, on rare occasions the transposon disrupts a gene that fights cancer. When this happens the cell doesn't die, it mutates into a tumor cell. When that cell divides, the result is 2 identical cells that both have lost the ability to fight cancer. Then both of *those* cells divide and … well, you can do the math.

A tiny example: suppose a chromosome is ATCAGGGGG, where TC is the transposon and GGGGG is a sequence of instructions for anti-cancer defense. After transposition, the chromosome might be AAGGGGGCT, which would not be a problem; or it might be AAGGCTGGG, which would be a problem.

In order to understand transposons, we need to model them in software. Transposons are all about inserting and deleting. You have seen that for very long collections, inserting with a linked list can be significantly faster than inserting with an array list. The same is true for strings: manipulating a linked list is much faster than manipulating a string. To model transposons and battle cancer, we need to abandon array lists and strings, and write linked list code. The code you write today will be a little different from the code you saw in lecture, where each node had a "next" variable. Today's nodes will have both "next" and "previous" variables, and they will be chained together to form what are called *doubly-linked* lists. You will see how taking care of nodes is more work than with an ordinary (*singly-linked*) lists; the benefit is that certain operations on the lists, which would be hard or slow with a singly-linked list, are fast and easy with a doubly-linked list.



Setup

Create an Eclipse workspace and a Java project in the workspace. Create a package called linked. Load the 2 starter files Node.java, and DNALinkedList.java into your package.

Node.java

The Node class is like the examples you've seen in lecture, except it has a prev (short for "previous") instance variable in addition to next. Note that the data instance variable has type Object. This is a problem because when apps retrieve the data, it is necessary to cast the data to a more useful type. As you've seen, lots of things can go wrong with this approach, and generics were invented to make the compiler detect such problems before run time. So convert Node.java to be generic. Just follow the instructions in the comment at the top of the file.

DNALinkedList.java

This class uses the generic Node class. Note that every Node declaration is "Node<Character>" because every node's data is one of the characters A, C, G, or T.

The constructors for this class are already written. Your job is to finish writing 8 methods in this class: append(), matches(), find(), extract(), reverse(), insert(), transpose(), and main(). The comments in the methods will tell you what to do. Look for "???", which you will replace with beautifully designed code that is correct, efficient, and sustainable. As you work, be aware that whenever 2 nodes are linked together or disconnected, you need to take care of the "next" of one of the nodes, and the "prev" of the other node.

In order to complete the main() method, you will need to create your own test case. The code prompts you to specify a chromosome string, a transposon string, and a target string. These will be used to test the transpose() method, which calls all the others. One way to test would be to use a real chromosome. The shortest human chromosome is 48 million bases long. Scribe: why would that be a bad first test case?

Discuss what features a good first test case should have. Scribe: what are these? (Hint: one feature is that they should be short enough that you can draw the diagrams described below.) Now create the 3 strings chromosome, transposon, and target. Scribe: write the 3 strings. What do you expect the DNALinkedList to look like after the call to transpose()?

On a piece of paper, draw diagrams like the yellow "Doubly-linked list" diagram above, showing the expected structure of your test case list at various points in transpose():

- At the start of the method.
- After finding firstNodeOfTransposon, firstNodeOfTarget, and lastNodeOfTransposon. Draw arrows pointing to each of these nodes.
- After extracting the transposon (also draw the extracted transposon).
- After the transposon is reversed (only the transposon list will change).
- At the end of transpose(), after the reversed transposon has been inserted back into the chromosome.

Label your diagrams clearly and neatly, and show them to your lab instructor. If you have the technology to do so, take pictures of your diagrams and Scribe: paste them into your report.

Now test your code. If you don't see the expected result, look at the transpose() code. After every line that changes the list, add a System.out.println call that prints the list and maybe the transposon. Scribe: what is the earliest point where something doesn't look like t expected diagram? Your bug is probably just before that point, so start looking there. Now or later, for class participation credit, post to the lab9 folder on Piazza and tell us if the diagrams were helpful.

Lastly: look at the extract() and insert() methods. Discuss how much harder they would be to write, if the list were singly linked. Scribe: for each method, would implementation with a singly linked list be a little harder, a lot harder, or about the same? Write in clear complete sentences so your instructor can know what you're referring to.