Introduction to the Assignment

In this lab, you will create a program that simulates creation of recombinant DNA. You will be given a DNA sequence, an enzyme and a DNA sequence to splice into the original DNA, replacing all occurrences of the enzyme.

The objectives of this assignment are to become familiar with ArrayList and to gain more experience with subclassing and JUnit.

Background

DNA consists of a sequence of four nucleotides: adenine, cytosine, guanine and thymine, abbreviated as A, C, G and T, respectively. Human DNA contains approximately 3,000,000,000 nucleotides. The order of these nucleotides determine how an organism is built. See What is DNA? (http://ghr.nlm.nih.gov/handbook/basics/dna) for more details.

Recombinant DNA is DNA that is manufactured by taking one piece of DNA, cutting it with a restriction enzyme, and inserting a second piece of DNA into the original piece. Recombinant DNA is used to make crops drought and heat resistant and is being used to treat genetic diseases such as cystic fibrosis and sickle cell anemia. For more details, see the Restriction Enzymes video at DnaTube.com (http://www.dnatube.com/video/955/Restriction-Enzymes).

Many bioinformatics programs manipulate DNA. Since DNA consists of just 4 nucleotides, each identified with a single letter, DNA sequences are typically represented as Strings, where the only characters present are A, C, G and T (or the lowercase equivalents). Many simulations can be carried out by doing string manipulation. In this lab, you will experiment with two representations of DNA: Strings and ArrayLists of Strings.

In the String representation, you will again use string manipulation operations to find enzymes and break the original DNA string into fragments. In this case, though, each fragment will be kept in a single element of the ArrayList. The new pieces of DNA will be inserted between the fragments.

For example, suppose the original DNA sequence is "ACGTGAGTGG", the enzyme used for splitting is "GTG" and the DNA sequence to splice in is "AAAA". In this case, there are 2 occurrences of the enzyme in the original DNA sequence. Each will be replaced with "AAAA". The final recombinant DNA is "ACAAAAAAAAG".

In the ArrayList representation, you will again use string manipulation operations to find enzymes and break the original DNA string into fragments. In this case, though, each fragment will be kept in a single element of the ArrayList. The new pieces of DNA will be inserted between the fragments.

Considering the same example as earlier, the initial DNA sequence will be an ArrayList with a single element:

"ACGTGAGTGG"

The final recombinant DNA will be an ArrayList containing several elements:

| "AC" | "AAAA" | "A" | "AAAA" | "G" |

Notice that the recombinant DNA sequence can be read by walking the ArrayList and concatenating the strings together.
The Program Design

This program performs a simulation, that is, the behavior of the program replicates something that happens in the real world. Simulations are often used by scientists to explore a question. In some cases, these simulations help a scientist to decide which physical experiments to try. In other cases, they completely replace physical experimentation, as, for example, in climate change research, where it is not generally possible to experiment with the climate! Such simulations are also often called “in-silico experiments”.

In this program, the in-silico experimentation only touches on what one might do to work with recombinant DNA. It does not answer any deep biology questions. Instead, from a computer science point of view, we can view this as an experiment to understand the relative performance of Strings and ArrayLists. For this reason, we will have three implementations of DNA strands. In the 2nd part of this lab, we will carefully study their performance.

This project has 1 abstract class and 3 classes that extend the abstract class, all in the dna package:

- **DnaStrand**: This is an abstract class that defines methods needed to create recombinant DNA and for the benchmark that we work on next week to gather information about the strand.
- **SlowStringDNAStrand**: This implementation keeps the DNA in a single string and uses only String methods to manipulate the DNA.
- **FastStringDNAStrand**: This implementation is similar to SlowStringDNAStrand, except that it uses a StringBuilder to manipulate the DNA.
- **ArrayListDNAStrand**: This is an implementation of a DNA strand where the DNA sequence may be broken across multiple elements in an ArrayList.

Doing the Lab

Step 1: Code the DNAstrand abstract class

Each of the classes that you will write must extend the DNAstrand class. DNAstrand should contain an instance variable that holds the original sequence in a String, a cutAndSplice method that does the recombination, an abstract append method and an abstract size method. Each subclass should also implement the toString method to return the recombinant DNA as a single string.

You can find a starter version of this class on the course website.

The main job of the DNAstrand class is to define the cutAndSplice method that is inherited by all the subclasses. This method must satisfy the following contract:

```java
/**
 * Cut the original DNA strand at every occurrence of enzyme, replacing every occurrence of enzyme with splicee. The original DNA sequence is not changed.
 * @param enzyme is the pattern/strand searched for and replaced
 * @param splicee is the pattern/strand replacing each occurrence of enzyme
 */
public void cutAndSplice(String enzyme, String splicee)
```

Use String’s indexOf and substring methods to find and extract pieces of DNA to put in your recombinant DNA. You may find it helpful to refer to the WebBrowser example from March 7 in which we extracted links from a Web page. When you want to add a piece of DNA to the recombinant DNA, call the append method that is declared in DNAstrand and that will be implemented in each of the subclasses.
Step 2: Code the SlowStringDNAstrand

For this class, store the recombinant DNA sequence as a String. Define the append method in SlowStringDNAstrand to concatenate the new DNA snippet onto the recombinant sequence. The size method should just return the length of the recombinant string. You should also override the default toString method to return the recombinant DNA string. The signature for toString is:

```java
public String toString()
```

If cutAndSplice is written correctly, you should find that the methods in SlowStringDNAstrand are all very simple.

Step 3: Test the SlowStringDNAstrand

Create a JUnit test for SlowStringDNAstrand. Be careful to pick good test cases so that you execute each line of code at least once, each condition taking both true and false branches. Also, make sure your loops work, even if they are executed 0 times. Also, keep in mind boundary cases. What happens if the enzyme starts the sequence, ends the sequence, or the only thing in the sequence is a single occurrence of the enzyme? What if the splicee contains an instance of the enzyme within it?

Step 4: Code the FastStringDNAstrand

The FastStringDNAstrand class should store the recombinant DNA in a StringBuilder object rather than a String. Look up StringBuilder in the Java API to find out how to add something to the end of a StringBuilder and to determine the StringBuilder’s size. You will also need to define a toString method for this class. Again, you should find the methods that you write in this class to be very simple.

Step 5: Test the FastStringDNAstrand

Create a JUnit test for this class. You will probably want very similar test cases as for the FastStringDNAstrands.

Step 6: Code the ArrayListDNAstrand

Next, you will code ArrayListDNAstrand. In this case, the DNA Sequence should be maintained as an array list where every element contains a piece of the DNA. Everytime that you append a DNA snippet to the recombinant DNA, put it in a separate element of the ArrayList.

You will again need to define append, size and toString. In this case, the methods are slightly more complicated than in the other two implementations, but not much.

Step 7: Test the ArrayListDNAstrand

Finally, create a JUnit test class for this class and test it thoroughly.

Grading

<table>
<thead>
<tr>
<th>Points</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Step 1: Code DNAstrand</td>
</tr>
<tr>
<td>5</td>
<td>Step 2: Code SlowStringDNAstrand</td>
</tr>
<tr>
<td>10</td>
<td>Step 3: Test SlowStringDNAstrand</td>
</tr>
<tr>
<td>5</td>
<td>Step 4: Code FastStringDNAstrand</td>
</tr>
<tr>
<td>5</td>
<td>Step 5: Test FastStringDNAstrand</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>10</td>
<td><strong>Step 6:</strong> Code ArrayListDNAStrand</td>
</tr>
<tr>
<td>5</td>
<td><strong>Step 7:</strong> Test ArrayListDNAStrand</td>
</tr>
<tr>
<td>20</td>
<td>Comments (including javadoc comments)</td>
</tr>
<tr>
<td>20</td>
<td>Style</td>
</tr>
</tbody>
</table>

**Credits**

This lab was inspired by a Nifty Assignment by Owen Astrachan of Duke University.