Com S 227
Fall 2017
Miniassignment 1
50 points

Due Date: Monday, October 16, 11:59 pm (midnight)
"Late" deadline (25% penalty): Tuesday, October 17, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html#ad **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Blackboard. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly, you will receive at most half credit.

Overview

The purpose of this miniassignment is to give you lots of practice writing loops and working with strings.

For this miniassignment you will implement one class, called `DNASequence`, that is a simple model of a strand of DNA. For our purposes, a DNA sequence is essentially a sequence of characters A, G, C, or T. We say that character A is the *complement* of T and the character G is the complement of C. One character forms a *bond* (a "base pair") with another when it is aligned side-by-side with its complement.

Given two DNA sequences, they tend to align with each other so that the number of bonds is maximized. Here's an example. Suppose we start with the sequence represented by the string

T C A T

and we want to examine possible alignments with another DNA sequence,

A G A G C A T

If we align them as

```
T C A T
A G A G C A T
```

then the number of bonds is two (T with A, and C with G):

```
T C A T
A G A G C A T
```

But if we shift the second sequence one space to the right,

```
T C A T
  A G A G C A T
```

then the number of bonds is just one (T with A).

If we shift two spaces to the right,

```
T C A T
    A G A G C A T
```

then there are no bonds, and if we shift three to the right,

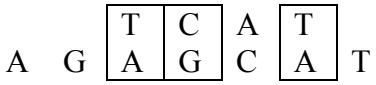
```
T C A T
      A G A G C A T
```

there is one bond (T with A).

We could also shift the second sequence to the *left*:

```
  T C A T
A G A G C A T
```

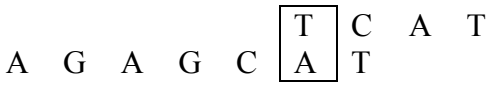
where there are no bonds. If we shift left by 2,



there are *three* bonds. We could also try shifting left by 3,

in which case there are no bonds, or by 4,

with one bond; by 5,

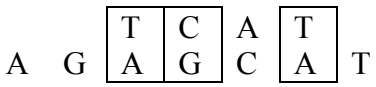


with one bond; or by 6,

where there are no bonds.

Thus the maximum possible number of bonds, for *all* possible alignments of the two sequences, is 3 (which occurs when the second sequence is shifted left by 2).

It is also useful to identify the characters at which the bonds occur. One possible notation would be to represent the unmatched letters as dashes. For example, with the given string TCAT and the alignment



we could represent the matches formed in TCAT using the string "TC-T".

Specification

The specification for this asssignment includes

- this pdf,
- any "official" clarifications posted on Piazza, and
- the online javadoc:

http://web.cs.iastate.edu/~cs227/homework/mini1/doc/mini1/DNASequence.html

The skeleton code

There is a complete skeleton for the **`DNASequence`** class linked on Piazza. It includes a declaration for one instance variable **`data`** of type String, which is initialized in the constructor and returned by the **`toString`** method. You are not required to implement it this way, but this is the simplest approach. You really should not need any additional instance variables.

To get started, create a new, empty project; within it create a package mini1, and within it create a class **`DNASequence`**. You can copy/paste the sample code into this class. It should compile without errors.

If strings are immutable, how do I ever change the data?

You can make a new string and assign it to the instance variable.

```
String temp = ... // do some work to make a new String
data = temp;
```

To modify or remove characters in a string, you can just iterate over it while making a new string that has the characters you want. For example, you can remove the vowels from a string like this:

```
String s = "cauliflower";
String temp = "";
for (int i = 0; i < s.length(); i += 1)
{
  char ch = s.charAt(i);
  if ("AEIOUaeiou".indexOf(ch) < 0)  // if ch is not a vowel
  {
    temp += ch;
  }
}
// now temp is "clflwr"
```

It's also possible to use the class `java.util.StringBuilder`, which is like a mutable form of `String`.

```
String s = "cauliflower";
StringBuilder sb = new StringBuilder();
for (int i = 0; i < s.length(); i += 1)
{
  char ch = s.charAt(i);
  if ("AEIOUaeiou".indexOf(ch) < 0)
  {
    sb.append(ch);
  }
}
String temp = sb.toString(); // now temp is "clflwr"
System.out.println(temp);
```

(The code isn't any simpler, but most Java programmers would use the second version because it's slightly more efficient. In the first version, the Java compiler is actually creating a `StringBuilder` behind the scenes in each iteration.)

You might be tempted to use the `StringBuilder` method `deleteCharAt` for the above problem of removing vowels. That is actually a bad idea - it's surprisingly tricky to delete elements from the sequence of things you're iterating over.

Advice

Start by doing simple, concrete examples with pencil and paper. Try to describe the steps of your solution in words, e.g. explain it to your little sister, explain it to your mom, explain it to your dog, until it makes sense to everyone. Then write it down in pseudocode. Be sure to give the dog a cookie.

When a problem seems difficult, start with a simpler, related problem. For example, counting bonds might have some tricky cases, but you can start with simpler ones:

- Given a string, can you iterate over it and just print out the characters?
- Given two strings the same length, like "TCATT" and "AGCGA", can you iterate over them and count the bonds for corresponding characters?
- Given two strings of *different* lengths, like "TCATT" and "AGCGAGT", can you determine the number of overlapping positions? Iterate over just the overlapping part?
- Given a string s of length 5 and a string t of length 7, if t is shifted two characters to the right, how many positions overlap? What are the minimum and maximum indices in s? How about in t?
- What if s is the longer one?
- What if you are shifting t to the left instead?

Similarly, finding the maximum number of possible bonds may seem hard, but you can start with simpler problems:

- Given two strings s and t, what is the largest amount that t could be shifted to the right and still overlap with s? What's the largest amount that t could be shifted to the left?
- How would you keep track of the maximum of a sequence of numbers that are being given to you one at a time?
- Can you find the maximum possible bonds, looking only at shifting to the right?
- Can you find the maximum possible bonds, looking only at shifting to the left?

The method `isSubsequence` may seem tricky too. Again, start by doing concrete examples by hand. You can try some related but simpler problems:

- Given two strings s and t, does every character of t also occur somewhere in s?
- Given a string s and character c, can you find the first index at which c occurs in s?
- Given a string s, character c, and integer pos, can you determine whether c occurs in s *after* position pos? Can you find its index in s?

Work incrementally and test as you go. Each simple example you do by hand can be turned into a test case. And if you get stuck on a small step, you'll have a simple, focused question to ask.

Getting started

1. As in previous assignments, create a new, empty project and add a package called `mini1`. Create the class `DNASequence` and copy/paste the skeleton code.

2. Method `willBond` is pretty easy to start with, and will be useful in many of the other methods. You don't even need a loop.

3. You could try `allValid`, which is kind of a warm-up problem, and is not a tricky loop to write.

4. The methods `complement()` and `fix()` are independent of everything else, and might be good as warm-up problems too. Be sure you have read the previous section "If strings are immutable...".

5. You'd probably want to work on **`countBondsWithShift`** before the rest. Read the "Advice" section above. Work incrementally and test things as you go. For example, you could implement the method incrementally so that it works...

- when the two strings are the same length and not shifted, then
- when the 'other' string is longer, but still not shifted, then
- when the 'other' string is shorter, but not shifted, then
- when the 'other' string is shifted right, then
- when the 'other' string is shifted left (this is likely to be a separate case)

6. The method **`findBondsWithShift`** is very similar to **`countBondsWithShift`**. It just requires you to construct a string to return instead of counting. Be sure you have read the previous section "If strings are immutable..." so that you know how to create a string in a loop (or use **`StringBuilder`**).

7. The method **`findMaxPossibleBonds`** depends on having correctly implemented **`countBondsWithShift`**. See the "Advice" section above.

8. The method **`isSubsequence`** may be the trickiest, but nothing else depends on it. Again,

I keep getting StringIndexOutOfBoundsException

Hmm. Could it be that you have a string index that is out of bounds? Remember, you are in complete control over what your code does. *If something hurts, stop doing it*!

Your first clue is in the exception message that appears in the console, which shows you exactly which line of code is causing the exception. Start reading at the top. When you see a reference to your own code, click on the link and it will take you to the line in your code where the exception occurred. That may be enough for you to spot the error.

If you don't immediately spot the error, take a simple test case and work through it by hand, to be sure you know what you *want* the code to do. Keep careful track of all the indices. Then trace by hand what your code is really doing.

You can also use the debugger to step through the execution. There are some exercises in this week's lab, including a demo of a nice feature of the Eclipse debugger called "exception breakpoints" with which you can have the code stop execution just *before* an exception occurs, so you can examine the state of all the variables before the program crashes.

For details, see this page of Lab 6: http://web.cs.iastate.edu/~cs227/labs/lab6/page08.html

The SpecChecker

Import and run the SpecChecker just as you practiced in Labs 1 and 2. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the console output. There are many test cases so there may be an overwhelming number of error messages. *Always start reading the errors at the top and make incremental corrections in the code to fix them.*

When you are happy with your results, click "Yes" at the dialog to create the zip file.

See the document “SpecChecker HOWTO”, which can be found in the Piazza pinned messages under “Syllabus, office hours, useful links” if you are not sure what to do.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, there are no specific documentation and style requirements.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`miniassignment1`**. If you don’t find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`miniassignment1`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled “pre” to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post “private” so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form “read all my code and tell me what’s wrong with it” will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled “Official Clarification” are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Blackboard, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_mini1.zip`**. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`mini1`**, which in turn contains one file, **`DNASequence.java`**.

Submit the zip file to Blackboard using the Miniassignment 1 submission link and verify that your submission was successful by checking your submission history page. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under “Syllabus, office hours, useful links.”

We strongly recommend that you just submit the zip file created by the specchecker. If you mess something up and we have to run your code manually, you will receive at most half the points.

> We strongly recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **mini1**, which in turn should contain the file **DNASequence.java**. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and not a third-party installation of WinRAR, 7-zip, or Winzip.