Assignment 2

Objectives

- Further experience working with Objects and Arrays in Java
- Experience writing your own tests
- Experience implementing and Interface in Java
- Experience working with command-line arguments

Objectives

This assignment requires you to implement an interface and then test your implementation.

Although there is a tester provided for this assignment, it does not include a comprehensive set of sets for each method. You should add your own tests to test cases not considered.

Note: The automated grading of your assignment will include different and additional tests to those found in the A2Tester.java file.

Quick Start

1. Download all of the .java files found in the Resources > Assignments > a2 folder.
2. Compile and run `A2Tester.java`
3. Read the directions beginning with the *Setting the Stage* section on the next page to determine how the implement the methods specified in the Aligner interface.
4. (Optional) Examine `SimpleAligner.java` and then compile and run `SimpleTester.java` to view the implementation and tests for a simple, almost-correct, implementation of the Aligner interface.

CRITICAL: You **must** name the methods in A2Aligner.java as specified in the documentation and used in `A2Tester.java` or you will receive a **zero grade.** Remember that all methods specified in an interface must be implemented, or your code will not compile correctly. Any compile or runtime errors will result in a **zero grade** (as if the tester crashes it will not be able to award you any points for any previous tests that may have passed).

Submission and Grading

Submit **A2Aligner.java** and **A2Tester.java** with your name and student ID at the top of each file using conneX**.**

If you chose not to complete some of the methods required, you **must provide a stub for the incomplete method(s)** in order for our tester to compile. If you submit files that do not compile with our tester, you will receive a **zero grade** for the assignment. It is your responsibility to ensure you follow the specification and submit the correct files. Additionally, your code must **not** be written to specifically pass the test cases in the tester, instead, it must work on all valid inputs. We may change the input values when we run the tests and we will inspect your code for hard-coded solutions.

Be sure you submit your assignment, not just save a draft. **ALL late** and **incorrect** submissions will be given a **ZERO** grade. A reminder that it is OK to talk about your assignment with your classmates, but not to share code electronically or visually (on a display screen or paper). We will be using plagiarism detection software.

Setting the Stage

An important task in bioinformatics is the alignment of DNA and RNA sequences. For our purposes a *nucleic acid sequence* (or *sequence* for short) is a string of letters: *A* for *adenine*, *C* for *cytosine*, *G* for *guanine*, and *T* for *thymine*. Here is one possible string:

```
TATGCCTCCGGTACATCAAC
```

In a *pairwise sequence alignment*, two such sequences are compared in order to determine how best one aligns with the other such that the nucleic acid sequences match each other. Real sequence data as used by biochemists and bioinformatics research consist of very long strings of A, C, G and T. Determining alignment for the general case can require the use of complex algorithms.

For this assignment, we will do something much simpler. We will attempt to align two sequences where the second string, we will call the subsequence, is shorter than the first. Here is an example of a subsequence:

```
TCAA
```

In aligning the subsequence to the first sequence above, we try to find the character position in the first sequence (or *offset*) at which the characters of the subsequence also appear. In our example, that offset would be 15 (an offset of 0 would mean the beginning of the sequence is identical to the subsequence). We could visualize the alignment of the sequences as follows (where periods are used as filler characters):

```
TATGCCTCCGGTACATCAAC
...............TCAA.
```

Unfortunately, we cannot always expect the subsequence to appear exactly at the beginning of the larger sequence. Sequence alignment is made more difficult as the alignment must take into account the comparison of each base along the sequence. For example, consider the following alignment:

```
ACGGTACGGAGTCTTGAAGT
GTTA
```

The subsequence GTTA does not appear anywhere in the sequence above. However, we still wish to attempt the best possible alignment. As a thought experiment, imagine we try to align the two sequences by using an offset of 0. This would give us:

ACGGTACGGAGTCTTGAAGT
~~GTTA~~................

Unfortunately this is a poor alignment. No characters match between the sequences (as indicated by the use of red-coloured text and ~~strikethroughs~~. Using an offset of one (1) is no better:

ACGGTACGGAGTCTTGAAGT
.~~GTTA~~...............

However, if we use an offset of two (2) then the situation improves:

ACGGTACGGAGTCTTGAAGT
..G~~T~~TA..............

In the first two attempted alignments we have four *errors* (i.e., mismatched characters between the strings). In the third attempted alignment we have one *error*. We slightly change our visualization by using an "x" in the second string for a mismatched character:

```
ACGGTACGGAGTCTTGAAGT
..GxTA..............
```

We now restate our goal for alignment: to find the leftmost offset where the number of *errors* is minimized. In the example given with the sequences above the best offset happens to be 2 (with just a single error).

Consider one last example, using the same longer sequence, but a different second string, `CAGT`. Here we see there are 3 offsets where there is exactly one error. For this assignment, if a tie occurs, the leftmost offset will be chosen (offset 1 in the following example):

```
ACGGTACGGAGTCTTGAAGT
.CxGT...xAGT....xAGT
```

The Aligner interface

The `Aligner` interface has documentation for the main two methods you will implement in `A2Aligner.java`. The `SimpleAligner.java`, provided for you, is an implementation of this interface.

SimpleAligner is not a correct implementation; it only compares sequences with an offset of 0, and does not correctly count the number of errors (unless there are no errors at all). So, it is useful and accurate only if the subsequence has 0 errors with an offset of 0. It may be useful for you to to see how it implements the `Aligner` interface, and also how it is tested (in `SimpleTester.java`).

The getUserInput method

After you have completed your implementation of the `Aligner` interface in `A2Aligner.java` there is one last method you must complete in `A2Tester.java`.

The `getUserInput` method can be used to perform sequence alignment tests quickly and easily by inputting the test sequences as command-line arguments when executing your program.

Please watch which overviews how to use command-line arguments in Java (may be prompted to login to ConneX first): https://connex.csc.uvic.ca/access/content/group/6eb1a305-4430-415c-95dd-5907babf80fd/videos/html/command-line-args.html

Using command-line arguments allows us to test any sequences quickly and easily, for example executing our program the following way:

```
java A2Tester CAGATACCATAGGGCATGC ATAGC
```

passes `CAGATACCATAGGGCATGC` and `ATAGC` as command-line arguments to our program. The `args` array passed as a parameter to main will now have the following contents:

```
String[] args = {"CAGATACCATAGGGCATGC", "ATAGC"}
```

The getUserInput method should first check if the command-line arguments passed in are valid (ie., there are 2 of them, the first String should be a large sequence, and the second should be a smaller String representing the subsequence to align with the first).

Afterwards, the getUserInput method should create a new A2Aligner object and determine the best alignment of the two sequences. We see below that the best alignment for the given command-line arguments is an offset of 3, as only a single error is found:

```
CAGATACCATAGGGCATGC
...ATAxC...........
```

If the command-line arguments are invalid, a new A2Aligner object is returned with sequence set to "invalid". This object is returned in the current implementation of the method provided for you.

Using the command-line arguments can be a quick and easy to test your A2Aligner.java implementation with many different inputs. (In fact, you could perform String alignments with Strings that don't only include A, C, T, and G if you are tired of nucleic acid sequence pairwise alignments). Have fun!