COMPUTER SCIENCE 21A (SUMMER, 2015)
DATA STRUCTURES AND ALGORITHMS

PROGRAMMING ASSIGNMENT 1

Due Wed, June 17 @ 10:00pm

- Submit assignment via Latte

PART ONE: ALL HANDS ON DEQUE

A **double-ended queue** or **deque** (pronounced "deck") is like a queue, except that it supports adding and removing items from either the front or the back of the data structure. Rather than the terms *enqueue* and *dequeue*, the terms used in the literature are *addFront*, *addRear*, *removeFront*, and *removeRear*. Create a generic data structure Deque that implements the following API:

```
public class Deque<Item> implements Iterable<Item> {
        // construct an empty deque
        public Deque(){}
        // is the deque empty?
        public boolean isEmpty(){}
        // return the number of items on the deque
        public int size(){}
        // add the item to the front
        public void addFront(Item item){}
        // add the item to the end
        public void addRear(Item item){}
        // remove and return the item from the front
        public Item removeFront(){}
        // remove and return the item from the end
        public Item removeRear(){}
        // returns a string showing the state of the deque
        public String toString(){}
        // return an iterator over items in order from front to end
        public Iterator<Item> iterator(){}
}
```

Corner Cases

There are a number of cases that you should handle consistently in your implementation:

- Throw a java.lang.NullPointerException if the client attempts to add a null item
- Throw a java.util.NoSuchElementException if the client attempts to remove an item from an empty deque
- Throw a java.lang.UnsupportedOperationException if the client calls the remove() method in the iterator (in other words, an iterator constructed by the iterator() method need not be capable of removing items)

- Throw a java.util.NoSuchElementException if the client calls the next() method in the iterator and there are no more items to return.

Blacklisted Classes
For this assignment, you may not use any elements from the java collections framework (Lists/ArrayLists, Trees, Maps). If you choose to use arrays, they must still scale linearly with the size of the input.

Performance requirements
Your deque implementation must support each deque operation (including construction) in *constant worst-case time* and use space linear in the number of items *currently* in the deque. Additionally, your iterator implementation must support each operation (including construction) in *constant worst-case time*.

A problem you will almost certainly have
Make sure your importing the class you want to into your test/client files (for example, importing java.util.Deque will give you nightmares and troubles, watch out for Eclipse auto-imports!)

PART TWO: BUILDING BLOCKS OF LIFE

DNA (the genetic information carrier in all living things) is a double helix comprised of nucleotide pairs. The molecule Adenine (A), bonds with Thymine (T), and the molecule Cytozine (C) binds with Guanine (G). DNA stores genetic information in the sequence of A's, G's, C's, and T's that comprise your genetic code. Though these words sound intimidating, one can really just think of DNA as a large sequence of numbers in base four. That should make you think about DNA as a type of information storage (which it is). Today we are going to model this data-storage-molecule with a data-structure, which, rather than storing its information in a unified location, will use a series of linkages to encode its information and structure, much like the way you have learned about linked lists.

The General structure of a DNA molecule is comprised of two strands. Each has a 'three-end', and a 'five-end'. You can think of these as the "front" of the strand, and the "end" of the strand. An interesting thing is that these strands run opposite one another. Below is a diagram showing how one can visualize this, in a molecule of DNA. We always refer to a STRAND as a single "list" of bases, while a MOLECULE of DNA is comprised of two, interlocking, strands.

```
(3 End) A=C=T=G=C=C=G=T=C=T=C=A=A=T=A=G=A=C (5 End)   <- Strand 1
        | | | | | | | | | | | | | | | | | |
[Interchangeable]
(5 End) T=G=A=C=G=G=C=A=G=A=G=T=T=A=T=C=T=G (3 End)   <- Strand 2
```

However, there is slightly more nuance to it. A molecule of DNA doesn't necessarily have "even" ends. Sometimes one end is longer than the other. Remember that A only bonds to T, and C only bonds to G. This will become important in a moment. The following is also a valid molecule of DNA.

```
(3 End) A=C=T=G=C=C=G=T=C=T=C=A=A=T=A=G (5 End)
                | | | | | | | | | | | | | |
     (5 End)  C=G=G=C=A=G=A=G=T=T=A=T=C=T=G (3 End)
```

Uneven strand lengths are interesting because they allow us to "stitch together" two different pieces of compatible DNA (though not all DNA will be compatible). For example, if I had the following two molecules of DNA, I could combine them into the first example molecule. Note that the (3) end must pair with the (5) end, so as to create a new strand which still has a (3) end and a (5) end.

```
(3) A=C=T=G=C=C=G=T (5)        (3) C=T=C=A=A=T=A=G=A=C (5)
    | | | | | |                    | | | | | | | | | |
(5) T=G=A=C=G (5)        (3) G=C=A=G=A=G=T=T=A=T=C=T=G (3)
```

Programming

Your task (though cloaked in lots of scientific terminology), is actually not too difficult, and only involves three classes. We have written some of the DNAMolecule class and the DNAStrand for you, but you must write the entire nucleotide class. Though a detailed explanation of the expectations of each method can be found in the java-doc comments of each file, you are expected to support the following methods (using the Nucleotide class as your physical manipulation location, no arrays or collections allowed in this part of the assignment):

<u>DNAMolecules</u>
- **Construct a new, non-ragged, DNAMolecule** with a given sequence for one of the strands. The Sequence will be given in the 3 to 5 order.
- **Duplicate an existing DNA molecule into two new DNA molecules**. The process for completing this part must produce two new molecules, each with one of the original strands from your first molecule. Thus, each resulting DNA Molecule should contain one old and one new strand.
- **Try To Join Another DNA Molecule**. If possible, join this molecule with a given one.
- Extra Credit: Apply a restriction enzyme: Look for a sequence (in the ascending, 3 to 5 direction) on both strands, and cut it into two strands if the search sequence is found. (You only need cut once if it is found anywhere). Return the new DNA molecule, and modify the existing one.

<u>DNAStrand</u>
- **Construct a DNA strand** from a given Character sequence.
- **Zip Together**: Takes two distinct DNA Strands, and interconnects them at a given offset, so that each nucleotide is now aware of its partner on the other strand.

Nucleotide

- Maintains connections to the next nucleotide, the previous nucleotide, and the paired nucleotide (on the opposite strand). The implementation of this class is entirely up to you.

Extra Credit: Restriction Enzymes

Another interesting operation over DNA molecules is the use of restriction enzymes, which you can think of as a pair of scissors. A restriction Enzyme (as we are defining it), is tasked with looking for a specific sequence of nucleotide bases ("CGACA", for example). If it finds that sequence (looking from the 3 end to the 5 end of BOTH STRANDS of the molecule), then it "cuts" the DNA into two new molecules of DNA. A restriction enzyme always produces a ragged cut, and the ragged cut is the search sequence. For example, if we take our original DNA molecule,

```
(3 End) A=C=T=G=C=C=G=T=C=T=C=A=A=T=A=G=A=C (5 End)
        | | | | | | | | | | | | | | | | | |
(5 End) T=G=A=C=G=G=C=A=G=A=G=T=T=A=T=C=T=G (3 End)
```

And we apply a restriction enzyme AATA on it, it would split the original molecule into two molecules. The function would modify the original strand, and return a new DNAMolecule object with the new structure.

```
(3) A=C=T=G=C=C=G=T=C=T=C=A=A=T=A (5)      (3) G=A=C (5)
    | | | | | | | | | | |                       | | |
(5) T=G=A=C=G=G=C=A=G=A=G (3)      (5) T=T=A=T=C=T=G (3)
```

Your extra credit task (should you chose to accept it) would be to code up a method that acts as a restriction enzyme on a given molecule of DNA. If you choose to accept it, it should be a method of the DNAMolecule object.

For questions on any part of this assignment, please email the TA (gward@brandeis.edu)