Testing Document Five

4/20/16

Kameron Damaska

Contents

1 DoubleLinkedList Test

1.1 addToFront method

This method takes a single DLNode of a generic type and adds it to the end of the DoubleLinkedList of the same generic type.

The test must cover:

- adding several nodes to a DoubleLinkedList.
- traversing through the DoubleLinkedList
  - done from the front of the list and back of the list.
- comparing the expected element in each node to the actual element
- checking nodes beyond the expected length of the DoubleLinkedList. It is expected that this will return null.

1.2 addToBack method

This method takes a single DLNode of a generic type and adds it to the start of the DoubleLinkedList of the same generic type.

The test must cover:

- adding several nodes to a DoubleLinkedList
- traversing through the DoubleLinkedList
  - done from the front of the list and back of the list
- comparing the expected elemtn in each node to the actual element
- checking ndoes beyond the expected length of the DoubleLinkedList. It is expected that this will return null.

1.3 equals

This method overrides Object's equals method. It has one Object parameter and compares whether it equals this DoubleLinkedList.

To be equal, the Object must be a DoubleLinkedList of the same length with the same contents in each node.

The test must cover:

- A parameter that is not a DoubleLinkedList.

- A DoubleLinkedList parameter whose nodes contain a different type from this DoubleLinkedList
- A DoubleLinkedList parameter whose nodes contain the same type as this DoubleLinkedList, but they differ in length
- A DoubleLinkedList parameter whose nodes contain the same type and is the same length as this DoubleLinkedList.
- Two empty DoubleLinkedLists

1.4 append

This method takes a DoubleLinkedList parameter and appends it to this DoubleLinkedList.

The test must cover:

- appending a DoubleLinkedList when this DoubleLinkedList is empty
- appending an empty DoubleLinkedList
- appending when neither DoubleLinkedList is empty

1.5 iterator

This method returns a ListIterator that starts at the head of the list and performs various methods on the list.

1.5.1 hasNext

Returns whether next can return an element

The test must cover:

- traversing forwards through a list
- testing when there is no element to return

1.5.2 hasPrevious

Returns whether previous can return an element

The test must cover:

- traversing reverse through a list
- testing when there is no element to return

1.5.3 next

Returns the element pointed to in the list and moves the pointer to the next

The test must cover:

- traversing forward through a list
- checking if the element returned at each next() is expected
- trying to return an element beyond the list's length

1.5.4 previous

Returns the element pointed to in the list and moves the pointer to the previous

The test must cover:

- traversing reverse through a list
- checking if the element returned at each previous() is expected
- trying to return an element beyond the list's length

1.5.5 add

Inserts a DLNode behind of the node currently pointed at. The element in the DLNode is the parameter from this method.

The test must cover:

- adding an element to an empty list
- adding an element at the end of the list
- adding an element at some point in the middle of the list
- adding an element at the start of the list

1.5.6 set

Sets the element last returned by either next() or previous() equal to the parameter from this method. If next() or previous() has not been used, or add() has been used since the last next() or previous(), then an error should be thrown.

The test must cover:

- setting an element after next() was used and see if the expected value was set

- setting an element after previous() was used and see if the expected value was set
- trying to set an element when neither next or previous has been used
- trying to set an element following the use of add()

2 DNA Test

2.1 toString method

Returns a String representation of a DNA. The String should be the letter representation of Bases (A, C, G, T) with no spaces between each letter.

The test must cover:

- A DNA with no Bases on it
- A DNA with only one Base on it
- A DNA with several Bases on it

2.2 string2DNA method

Takes a single String input and returns the DNA sequence that the String represents. To represent a DNA sequence, the String must not be empty, and its contents must represent a Base (A, C, G, or T).

The test must cover:

- Trying to convert an empty String to DNA
- Trying to convert a String that doesn't represent a DNA sequence to DNA
- Converting Strings of various lengths to DNA

2.3 splice method

Takes int and DNA parameters. The first n number of Bases of the DNA parameter are removed. The remainder is appended to the end of this DNA. The parameter can be destroyed when the int parameter is greater than the number of Bases on the DNA.

The test must cover:

- Appending DNA that got destroyed through the splicing. Both with the int parameter being equal to the DNA length, and greater than the DNA length.

- Appending DNA that has one Base on it following the splice.
- Appending to a DNA strand with no Bases

2.4 overlaps method

Takes one int and two DNA parameters. Returns true when the last n bases of the first DNA parameter perfectly match the first n bases of the second DNA parameter.

The test must cover:

- Two DNA parameters that are both empty with an int parameter 0.
- Two DNA parameters that are both empty with a non-zero input.
- Two identical DNA parameters of length one with an int parameter 1.
- Two identical DNA parameters of length one with an int parameter greater than
- Two DNA parameters that match at int parameter n, but not at n + 1

2.5 main method

This method takes two Strings that represent DNA sequences. It determines the greater overlap between the two (either start of the first, end of the second or start of the second, end of the first). Then the appropriate splicing is done to create the shortest possible DNA strand. If the inputs are not two Strings that represent DNA sequences, then a message will be printed to inform the user.

The test must cover:

- Two DNA parameters with no overlap
- Two DNA parameters that fully overlap each other
- Two DNA parameters with the overlap being the start of the first and the end of the second.
- Two DNA parameters with the overlap being the start of the second and the end of the first.
- Parameters that will inform the user that bad data was input
  - One DNA parameter
  - Three DNA parameters
  - Two non-DNA parameters

2.6 compareTo method

To make DNA comparable, the class implements the Comparable interface. The Comparable interface contains one method (compareTo) that needs to be overriden.

This method compares this DNA with the specified DNA for order. Returns an int. Positive integer when this object is greater than the specified DNA. Negative integer when this object is less than the specified DNA. Zero when this DNA is equal to the specified DNA.

For organization is done by length. Shorter DNA are considered less than longer DNA. If equal length, it compares alphabetical order.

The test must cover:

- Two zero length DNA, returns 0.
- Comparing a larger DNA to a shorter DNA
- Comparing a shorter DNA to a longer DNA
- Comparing equal length DNA with different alphabetical order
- Comparing equal length DNA with identical alphabetical order