Message class - writing a message to the messages file Part I - availableList empty

We keep using the following messages. Now we implement the method that writes messages and sets internal links appropriately so that messages can be read from the file later.

	data (RECORD_DATA_LEN bytes)	**next** (pointer to next record: 4 byte int)
0	Dear Susan, how are you today? I am going to try to write you more frequently ab	2
1	Hi Johnny, do you want to go to the movies tonight? I have free passes. Suzie	-1
2	out the material that we had left open last time. It seems that the entries that	3
3	you had included in the invoices contain some calculation errors. For example, t	4
4	he first item has a price of $39.75 and there were 5 items ordered. The resultin	6
5	To everyone at the sales department: This is a reminder that our monthly luncheo	8
6	g price should be $39.75 x 5 = $198.75 but you have it showing as $201.25, and t	7
7	he tax is calculated based on this figure of $201.25. Could you please review th	9
8	n will take place at Favourite Pizza, 275 Pepperoni Road. See you all there.	-1
9	e entire worksheet and email me a new copy. I truly appreciate it. Sincerely Pet	10
10	er, Inventory Manager.	-1

Message class

Add an access method

public void setText(String t) that sets **text** to **t**

Implement the following method in the Message class

public int writeToMessagesFile()

that writes an entire message to the messages file. The method writes the implicit parameter to the file using first any possible available records from the availableList. If there are no records available and there is still portions of the message to be written to the file, then the records are appended and the file grows.

The method does not access the messages file directly but indirectly through the methods of the Record class.

The method takes no parameters. It writes the message as records and sets the correct next links.

The method returns the record number of the first record of the message. For example, if the message is written using records 12, 3, 6, 15, and 16 then the method returns 12.

The algorithm deals with two separate cases: 1) availableList is empty and 2) availableList is not empty. When the first case happens, all new messages will be appended to the messages file. In the second case, the messages are broken and the available spaces within the messages file is reused. We start with the first case.

Pseudo-code is below

```
// pseudo-code when the availableList is empty

method to write the string s to the messages file
    while s is not a null string {
        if the messages file is full {
            next available spot is at the bottom of messages file
            if the message fits in one record {
                set a record with data from message
                write the record in APPEND mode
                set s to null string
            }
            else {
                set a record with the first RECORD_DATA_LEN chars of s and
                                                        the subsequent record number
                write the record in APPEND mode
                remove from s the first RECORD_DATA_LEN chars
            }
        }
        else {
            <...code when availableList is not empty...>
        }
    }
    return the number of first record where message was written
}
```

```
public int writeToMessagesFile() {
    declare s as String s and assign text it (this will be slightly modified later)
    declare recordNumber as int and initialize to -1   // current record
    declare nextRecordNumber as int and initialize to -1    // link to next record
    declare startRecordNumber as int and initialize to total records in messages file
                                                       // first record of message

    declare and instantiate record as an object of the Record class

    while s is not a null string {
        if availableList is empty {
            recordNumber = total records in messages file
            if s fits in one record {
                record.setData(s, Globals.END_OF_MESSAGE);
                record.writeToMessagesFile(recordNumber, Globals.APPEND)
                s = "";  // forces out of loop
            }
            else {
                nextRecordNumber = recordNumber + 1
                record.setData(s.substring(0, Globals.RECORD_DATA_LEN), nextRecordNumber)
                record.writeToMessagesFile(recordNumber, Globals.APPEND)
                s = remove from s the portion that has just been written to file
            }
        }
        else {
            <...code when availableList is not empty...>
        }
    }
    return startRecordNumber
}
```

EmailServer testing

For all tests be sure to:

1) Manually remove the messages file when starting with an empty file
2) Use the following template in the main class

```
main() {
     instantiate the availableList to an empty list
     declare and instantiate an object of the Message class, for example message

     open the messages file (use FileIO.openMessagesFile() method; do not open directly)

     if successful
          set the text private variable to a message of length as specified below
          write the message with the call message.writeToMessagesFile()
          close the messages file
     end if
}
```

3) check the expected results as given in the table below

Tests

These tests are for the first part of the code, where the availableList is empty

test	conditions	code being tested	expected results
1	1 message that fits in a single record, starting with an empty file	availableList is empty and message fits in one record	messages file is one record of 84 bytes next link set to -1
2	2 messages that will each fit on a single record, starting with an empty file	same as above but when the second message is written messages file is no longer empty	messages file is two records of 168 bytes both next links set to -1
3	1 message that requires 2 records starting with an empty file	available list is empty first portion of message will go in one record and the second portion in another	messages file is two records of 168 bytes first link is set to 1 and second link set to -1
4	1 message that requires 4 records starting with a file that already has messages in it	available list is empty and message requires multiple passes through the second loop to accommodate for a 4 record message	message file has grown by 4 records and exactly 4*84 bytes links set properly
5	reading at least two messages from given starting positions	readFromMessages() from the Message class	messages have been read correctly and are in legible format to be read by a user