P08 LIFO Inbox Reader

Overview

In this assignment, you are going to develop a simple inbox reader application. The messages stored in/uploaded to the inbox can be read according to the Last In First Out (LIFO) policy, only. Our inbox consists of two stacks (one stores the unread messages and the other stores the read messages). You are going to implement a linked stack as well as all the operations related to the management of our iterable inbox.

Learning Objectives

The goals of this assignment include (1) implementing, using, and testing an implementation of the generic interface StackADT using linked nodes, (2) implementing an iterator to iterate over a linked stack, (3) implementing the Iterable interface.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P8 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 11/15/2020**. Access to the pre-assignment quiz will be unavailable passing its deadline.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
15 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Additional Assignment Requirements and Notes

- The only import statement that you may include in your classes are `import java.util.EmptyStackException; import java.util.Iterator;` and `import java.util.NoSuchElementException;`
- You MUST NOT add any fields either instance or static, and any public methods either static or instance to your `MessageStack`, `MessageStackIterator`, and `Inbox` classes, other than those defined in this write-up.
- DO NOT make any change to the provided source files `StackADT.java`, `Message.java`, and `LinkedNode.java`. In addition, **DO NOT submit these provided files including `InboxReader.java` on gradescope**.
- All your test methods should be defined and implemented in your `InboxReaderTester.java`.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define private methods to help implement the different public methods defined in this write-up, if needed.
- In addition to the required test methods, we HIGHLY recommend (not require) that you develop your own unit tests (public static methods that return a boolean) to convince yourself of the correctness of every behavior implemented in your `MessageStack`, `MessageStackIterator`, and `Inbox` classes with respect to the specification provided in this write-up. Make sure to design the test scenarios for every method before starting its implementation. Make sure also to test all the special cases.
- All implemented methods including the overridden ones MUST have their own javadoc-style method headers, according to the CS300 Course Style Guide.

1 Getting Started

Start by creating a new Java Project in eclipse called P08 Inbox Reader, for instance. You have to ensure that your new project uses Java 11, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-11" within the new Java Project dialog box. Then, add these provided StackADT.java, Message.java, and LinkedNode.java to the src folder of your project. The provided `StackADT` interface represents the generic abstract data type stack. This interface will be implemented by two non-generic stack data structures in the next steps of this assignment. The `Message` class implements the type of elements which will be stored in these stacks. The `LinkedNode` generic class implements a linked node that can be used to implement any singly-linked list based data structure.

2 MessageStack Implementation and Testing

Our inbox reader reads the message stored in the inbox according to the Last In First Out scheduling policy. To enable this property, we are going to first implement a stack called `MessageStack` which stores elements of type Message. First, we invite you to read through the provided implementation of the `Message` and `LinkedNode` classes. Then, create a new class called `MessageStack` and implement it according to the following specification.

2.1 Notes related to the MessageStack class

- Your MessageStack class must implement this StackADT interface. You have to override all the methods defined in that interface. Notice that this StackADT is similar (but not identical!) to the version discussed in lecture.
- Notice that the `StackADT` interface is generic. But, your class `MessageStack` must **NOT BE GENERIC**.A `MessageStack` MUST contain only elements of type `Message`.
- The `MessageStack` class **MUST define only two fields**:
  - a **private instance field** called **top** of type $LinkedNode < Message >$ which refers to the top of the linked stack, and
  - a **private instance field** called **size** of type **int** which keeps track of the total number of Message objects stored in the stack.
- Use the provided `LinkedNode` class to implement the stack using a chain-of-linked-nodes approach. **DO NOT create an array or ArrayList based stack implementation.**
- You do not need to implement a constructor for the class MessageStack. The compiler will add a no-argument constructor which sets the top and the size of the stack to their default values (null and zero). The default no-argument constructor creates a new empty MessageStack.

2.2 Design and implement your unit tests first!

Now, we want you to write the test methods to specify and validate what the methods defined in the MessageStack will do. Then, you can implement these functionalities in your `MessageStack`. Starting by developing the test cases of each behavior will help writing a code which is easy to debug or bug-free. To help you in this process, we provide you with a starting code for your tester class. Download the class `InboxReaderTester` and start the implementation of the following two test methods.

```
public static boolean testStackConstructorIsEmptyPushPeek(){ }
public static boolean testStackPop(){}
```

We provided you with a set of test scenarios to consider in the above tester methods. If you do find bugs in your implementation in the future, you can add additional test scenarios to help you detect the bugs and fix them. It is also worth noting that before checking the correctness of the MessageStack.pop() method, you have to check the correctness of both .push() and .peek() methods first. This is simply because you have to call those both methods to build the content of the stack appropriately, and then double check that the element at the top of the stack has been correctly removed and returned when .pop() method is called.

3 Create the Inbox class

Now, create a class called `Inbox`. This class represents the inbox where all received/loaded unread and read message are stored and managed. Your `Inbox` class must have the following fields (you will not need any others).

```
private MessageStack readMessageBox; // stack which stores read messages
private MessageStack unreadMessageBox; // stack which stores unread messages
```

In addition, your `Inbox` class must implement all the following public methods including one no-argument constructor.

```
public Inbox() {
  // This no-argument constructor creates a new empty inbox
 // and initializes its instance fields.

 // Both unreadMessageBox and readMessageBox stacks of this
 // inbox must be initially empty.
}

public String readMessage() {
  // Reads the message at the top of the unreadMessageBox.
 // Once read, the message must be moved from the unreadMessageBox
  // to the readMessageBox.

 // This method returns the string representation of the message at
 // the top of the unreadMessageBox, or "Nothing in Unread"
 // if the unreadMessageBox of this inbox is empty.
}

public String peekReadMessage() {
 // Returns the string representation of the message at the top of the readMessageBox.

 // This method returns the string representation of the message at the top
 // readMessageBox and "Nothing in Read" if the readMessageBox is empty.
}
```

```
public int markAllMessagesAsRead() {
 // Marks all messages in the unread message box as read.
 // The unread message box must be empty after this method returns.
 // Every message marked read must be moved to the read messages box.

 // This method returns the total number of messages marked as read.
}

public void receiveMessage(Message newMessage) {
  // Pushes a newMessage into the unread message box
  // newMessage represents a reference to the received message
  // Note that this method can be invoked each time a new message
  // will be received and pushed to the unreadMessageBox.
}

public int emptyReadMessageBox() {
  // Removes permanently all the messages from the readMessageBox

  // This method returns the total number of the removed messages
}

public String getStatistics() {
  // Gets the statistics of this inbox
 // Returns a String formatted as follows:
 // "Unread (size1)" + "\n" + "Read (size2)",
 // where size1 and size2 represent the number of unread and read
 // messages respectively.
}
```

3.1 Expanding The InboxReaderTester Class

You are responsible for testing thoroughly your implementation of the Inbox public methods. We provided you with the method signature of `testInboxReadMessage()`, `testInboxReceiveMessage()`, and `testInboxMarkAllMessagesAsRead()` as examples of test methods that you can consider to check the correctness of the implementation of the Inbox class. Any test methods you add to your `InboxReaderTester` class should be **public static boolean** and take no parameters, returning true when they detect correct behavior of your program and false otherwise. You may add additional private helper methods as necessary. You may also add a main method to your testing class. In particular, you must add the following test method:

```
public static boolean runInboxReaderTestSuite() { }
```

Your `runInboxReaderTestSuite()` test method should run all the test methods implemented in your InboxReaderTester class. It **should return false if any of its component tests fail,**

and true if they all succeed. Note that we will run this specific test method against several `MessageStack` or `Inbox` implementations (some working, some broken) in our automated tests on Gradescope.

4 Create MessageStackIterator Class and Make Your MessageStack Class Iterable

You can notice that there is no way actually to traverse the unread and read messages in our inbox. They are stored in two stacks and the peek method only gets the message at the top of the stack. To provide the user with the ability to traverse these stacks, we are going first to implement an iterator to iterate over our MessageStack class. Create a new class called `MessageStackIterator` and add it to your project.

4.1 Notes related to the MessageStackIterator class

- Your `MessageStackIterator` class must implement $Iterator < Message >$ and must iterate directly over your stack.
- Your `MessageStackIterator` class must define a private instance field of type $LinkedNode < Message >$ to keep track of the next element in the iteration.
- The constructor of your iterator must take a $LinkedNode < Message >$ as only input parameter (which represents the top of the stack).
- MessageStackIterator's next() method should throw NoSuchElementException with a descriptive error message if the stack is exhausted and the current element in the iteration does not have a next item.
- Recall that when called for the first time, next() method defined in MessageStackIterator class must return the message at the top or head of the stack (if it is not empty).

4.2 Expanding The InboxReaderTester Class

Now, expand your tester class by implementing the `testMessageStackIterator()` unit test method. Hints about the scenarios that you can consider to check the correctness of the MessageStackIterator class are provided in the details of the javadoc style method header in the `InboxReaderTester` class.

4.3 Expanding The MessageStack Class

Now, let's make our MessageStack class iterable, so that we can use a for-each loop to traverse and explore its contents. In addition to the StackADT, your `MessageStack` class must implement

the java.lang.Iterable interface ($Iterable < Message >$). Your MessageStack's `iterator()` method must return a new MessageStackIterator that starts at the top of the stack of Messages.

4.4 Expanding The Inbox Class

Now, you can expand your `Inbox` class by adding and implementing the following two methods with the exact following signatures.

```
public String traverseUnreadMessages() {
  // Traverses all the unread messages and return a list of their
 // ID + " " + SUBJECT, as a string. Every string representation of a
 // message is provided in a new line.

 // This method returns a String representation of the contents of
 // the unread message box.
 // The returned output has the following format:
  // Unread(unreadMessageBox_size)\n + list of the messages in
  // unreadMessageBox (ID + " " + SUBJECT) each in a line.
}

public String traverseReadMessages() {
  // Traverses all the read messages and return a list of their string
 // representations, ID + " " + SUBJECT, each per new line, as a string

  // This method returns a String representation of the contents of
  // the read message box
  // The returned output has the following format:
  // Read(readMessageBox_size)\n + list of the messages in
  // readMessageBox (ID + " " + SUBJECT) each in a line.
}
```

5 Driver Application InboxReader Class

As the final step of this assignment, download and use this driver class called InboxReader.java to launch and run the Inbox Reader application and make use of the different classes implemented in this project. Read through the implementation of the provided InboxReader class to understand how it works. Running the main method in this class must result in an interactive session like the one demonstrated in the following sample output. You do not need to submit or make any change to the provided `InboxReader` class. Launching its main method should immediately begin a driver loop and present the user with a menu with different options to load the unread messages, and then read and manage the messages read or unread stored in the inbox. We used this messages.txt as input file to load the inbox (unread messages).

Sample Output of the driver application

```
--- Welcome to our LIFO Inbox Reader App! ----

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 8
Unread (0)
Read (0)

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 1 messages.txt
10 unread messages loaded to the inbox.

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
```

```
-------------------------------------------
ENTER COMMAND: 4
Unread(10)
10 Deadline Approaching
9 Positive Waves
8 COVID vaccine
7 Hints
6 Reminder
5 P02 Feedback
4 Greeting
3 Lecture
2 P01 Feedback
1 Hello

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 5
Read(0)

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 2
[10] Deadline Approaching: The deadline to complete p08 is on Wed 11/18

==================== MENU ====================
Enter one of the following options:
```

```
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 2
[9] Positive Waves: Smile, be positive, and keep going on!

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 2
[8] COVID vaccine: Horray! Covid-19 vaccine from Pfizer and BioNTech
showed a strong effectiveness (90%).

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 8
Unread (7)
Read (3)

==================== MENU ====================
```

```
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 3
[8] COVID vaccine: Horray! Covid-19 vaccine from Pfizer and BioNTech
showed a strong effectiveness (90%).

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 4
Unread(7)
7 Hints
6 Reminder
5 P02 Feedback
4 Greeting
3 Lecture
2 P01 Feedback
1 Hello

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
```

```
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 6
7 messages marked as read.

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 8
Unread (0)
Read (10)

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
-------------------------------------------
ENTER COMMAND: 2
Noting in Unread

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
```

```
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 3
[1] Hello: We appreciate you. Take care and stay safe.

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 7
Read Empty. 10 read messages deleted.

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 8
Unread (0)
Read (0)

==================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
```

```
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 3
Noting in Read

=================== MENU ====================
Enter one of the following options:
[1 <filename>] Load Inbox
[2] Read message
[3] Peek read message
[4] Print list of unread messages
[5] Print list of read messages
[6] Mark all messages as read
[7] Empty Read
[8] Print Statistics
[9] Logout and EXIT
---------------------------------------------
ENTER COMMAND: 9
------- BYE! Thanks for using our App! -------
```

6 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through gradescope.com. The only 4 files that you must submit include: `MessageStack.java`, `Inbox.java`, `MessageStackIterator.java`, and `InboxReaderTester.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline.