**Lab Assignment 7:**

**Stacks using a LinkedList**

|  |
| --- |
| **Due Date**  **(a two-week LA)** |
| **04/20/16 11:59pm – For Wednesday Lab Sections**  **04/21/16 11:59pm – For Thursday Lab Section** |

**Objectives**

* Working with Stacks
* Working with Linked Lists (singly-linked)
* Working with Binary Files.

**Problem Specification**

This assignment is a modification of LA6. You will be implementing your own version of a singly-linked list and Stack and then you will be using them to solve a simple programming problem. Note that a singly-linked list is made up of objects, each of which has data stored in it and in addition has a reference to the object that follows it in the list. Being a singly-linked list however, an object in the list does not know which object precedes it. In this implementation, you will be using nodes for the objects in the linked list. An INode interface is provided which is to be implemented by a Node class. The Node class in the Java API is NOT to be imported or used in this assignment.

**Problem to Solve:**

Given an input string, reverse the string. You must use your implementation of Stack in order to do this. Your must use your implementation of Linked list in order to store the file input.

**Input File**

A binary input file with strings in UTF-8 format will be provided, named **input.bin**.

Each input string will be provided on a single line.

Do not assume the file will have any specific length.

**Example Input File:**

The cat ate the rabbit.

yOU CAN’T COMPARE appLES TO oRANGEs.

**Output:**

You must print the reverse of each string into a binary file called output.bin in UTF-8.

You must also print the following to the screen:

The reverse of string “The cat ate the rabbit.” is “.tibbar eht eta tac ehT”

The reverse of string “yOU CAN’T COMPARE appLES TO oRANGEs.

” is “.sEGNARo OT SELppa ERAPMOC T’NAC UOy”

The interfaces to be implemented are provided below. The main class, which is **not to be modified**, is also provided.

**public** **interface** IList {

/\*\*

\* Adds the element e to the end of the list.

\* **@param** e element to be added

\*/

**void** add(String e);

/\*\*

\* Adds the element e to the end of the list.

\* **@param** index of the location to place the string, starting from 0

\* **@param** e element to be added

\*/

**void** add(**int** index, String e) **throws** IndexOutOfBoundsException;

/\*\*

\* Removes all of the elements from the list

\*/

**void** clear();

/\*\*

\* Checks to see if list contains the parameter s

\* **@param** s parameter to search for.

\* **@return** true if found, false otherwise.

\*/

**boolean** contains(String s);

/\*\*

\* **@return** the element at the front (index 0) of the list

\*/

String getHead();

/\*\*

\* **@return** the element at the end (index size-1) of the list.

\*/

String getTail();

/\*\*

\*

\* **@param** Index of the element to retrieve. (Indexing starts from 0.)

\* **@return** the element at that index.

\* **@throws** IndexOutOfBoundsException

\*/

String get(**int** index) **throws** IndexOutOfBoundsException;

/\*\*

\* Searches for the element s in the list and returns the

\* index of the first occurrence, starting from index 0

\* **@param** s parameter to search for

\* **@return** index of the element, or -1 if not found.

\*/

**int** indexOf(String s);

/\*\*

\* **@return** true if the list is empty, false otherwise.

\*/

**boolean** isEmpty();

/\*\*

\* Removes the element at the specified index.

\* **@param** Index of element to be removed. (Indexing starts from 0.)

\* **@return** The contents of the element that was removed.

\* **@throws** IndexOutOfBoundsException

\*/

String remove(**int** index) **throws** IndexOutOfBoundsException;

/\*\*

\*

\* **@return** the number of elements in this list.

\*/

**int** size();

}

Use the LinkedList class to implement the IList interface. The LinkedList class should use nodes to store its data. Each node should also have a reference to the node that follows it in the list. The LinkedList class should be 0 indexed (start index from 0).

**public** **interface** INode<E> {

/\*\*

\* Returns the data stored in this node.

\* **@return** Data in this node.

\*/

E getData();

/\*\*

\* Setter for data for this node.

\* **@param** data New data

\*/

**void** setData(E data);

/\*\*

\* Returns the node next to this node.

\* **@return** Node next to this node.

\*/

INode<E> getNext();

/\*\*

\* Sets node received as the next node to this node.

\* **@param** next New next node.

\*/

**void** setNext(INode<E> next);

}

The Node class should implement the INode interface.

**public** **interface** IStack {

/\*\*

\* Adds the parameter s to the top of the stack.

\* **@param** s the string to be added

\*/

**void** push(String s);

/\*\*

\* Removes the top element from the stack

\*/

**void** pop();

/\*\*

\* Returns the top element without removing it.

\* **@return** the top element in the stack

\*/

String peek();

/\*\*

\*

\* **@return** the number of elements in the stack

\*/

**int** size();

/\*\*

\*

\* **@return** true if the stack contains no elements, false otherwise.

\*/

**boolean** isEmpty();

}

Use the Stack class to implement the IStack interface above.

Your Stack class MUST use your IList implementation (singly-linked list).

**public** **interface** IApplication {

/\*\*

\* Reads the binary file "input.bin" and returns each line

\* as an element in an IList

\* **@return** an IList containing the input.

\*/

**public** IList readInputFile();

/\*\*

\* Writes the reversed string to the binary file "output.bin"

\* **@param** output

\*/

**public** **void** writeOutputFile(IList output);

/\*\*

\* Prints out the input and output strings to the screen.

\* @param input the input string

\* @param output the output string

\*/

**default** **public** **void** printToScreen(String input, String output){

System.***out***.println("The reverse of string \""+input+"\" is \""+output+"\".");

}

/\*\*

\* Reverses the String parameter.

\* **@param** s the String to be reversed

\* **@return** the reversed string

\*/

**public** String reverseString(String s);

}

Implement all methods in the Application class. Use the stack you implemented to help you reverse a string.

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Application app = **new** Application();

IList inputStrings = app.readInputFile();

inputStrings.add(2,"String added to index 2");

IList reversedStrings = **new** LinkedList();

**for**(**int** i=0; i<inputStrings.size();i++){

reversedStrings.add(app.reverseString(inputStrings.get(i)));

app.printToScreen(inputStrings.get(i), reversedStrings.get(i));

}

app.writeOutputFile(reversedStrings);

}

}

The main method/class has been provided for you. Use as is.

**Requirements:**

1. Create a project and name it edu.wmich.cs1120.SP16.LA7.YOUR\_LASTNAME
2. Create three packages:
   1. edu.wmich.cs1120.SP16.LA7.YOUR\_LASTNAME.application
   2. edu.wmich.cs1120.SP16.LA7.YOUR\_LASTNAME.lists
   3. edu.wmich.cs1120.SP16.LA7.YOUR\_LASTNAME.stacks
3. Add the IList interface to the lists package. Implement the interface in a class called LinkedList and add any extra methods as needed.
   1. Add the INode interface to the lists package and implement it in a class called Node.
   2. Do not use or import the built in LinkedList or Node classes.
4. Add the IStack interface to the stacks package and implement it.
   1. Your implementation must use your implementation of IList.
5. Add the application interface to the application package and implement it.
   1. Your reverseString function must use your implementation of IStack.
   2. The input file has been given to you. Make sure to read binary files properly.
6. Add the Main class to the application package. Use as is without modification.
7. Ensure that your program implements good exception handling.

**Additional Requirements**

**Coding Standards**

You must adhere to all conventions in the CS 1120 Java coding standard. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming classes, variables, method parameters and methods. You must use the package instructions included above.

**Teamwork**

You must work within your group on this assignment. You will be graded on how well you use your GIT repository. You must add your lab instructor to the GIT repository. Work must be divided equally between the partners (we will be to see your commits). If you procrastinate, we will know!

A zip file containing your submission only needs to be submitted to Elearning if you can’t get Git working (you will be penalized). If your code is in the repository, you will not have to upload anything to Elearning for the home portion.

**Assignment Submission Lab Portion**

1. **Lab Portion**

* A word or pdf document containing your UML diagrams plus pseudo code

1. **Home Portion**

* A zip file containing your LA6 project with Javadoc included.

**NOTE: Standards will be strictly enforced.**