CS-1120-Computer Science

Lab TA: Fola Olagbemi

Oliver Sanchez & Chris Tower

Lab Report LA 7

This is sample lab report for a List program, in which we manage a stack and file I/O by creating our own Linked List class and using a Node class.

**DESIGN PHASE**

1. **Basic Structure**

List all the methods you plan to have for this project. Give each method a one-sentence description explaining what this method does.

**METHODS & PSEUDOCODE OF CLASSES**

Class **Main**

**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.

Class **LinkedList** – implements IList interface

**void** add(String e);

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

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

{

if (isEmpty())

{

Add String e to array[0]

}

Else if

Check if array has space for data, if so add it

{

Make a new array of size of previous version of array +10

Then Add String e to array[array.lengthprevious + 1]

}

}

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

\* Adds the element e to the list at the specified index.

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

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

{

* If index is below zero, then throw exception

if (index < 0

{

String message = String.valueOf(index);

throw new IndexOutOfBoundsException(message);

}

* If index || index > size()
* Then need more space in array, add more space

// Index is at least 0

if (index == 0)

IF ARRAY IS EMPTY

**// New element goes at beginning**

Add data to beginning of array – POSITION ZERO

If not empty, check if index can be added to

Add date to index

**void** clear();

\* Removes all of the elements from the list

* Sets array to null
* Resets size to zero.

**boolean** contains(String s);

\* Checks to see if list contains the parameter s

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

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

* Boolean initially false
* Traverse array
* Set Boolean to true if element is found

String getHead();

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

Return first; // first is an attribute of this class.

String getTail();

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

Return last; // last is an attribute of this class.

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

\* **@param** index index of the element to retrieve, starting from 0.

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

\* **@throws** IndexOutOfBoundsException

**int** indexOf(String s);

\* 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.

- we have to iterate through the array to look for the index using a string.equals() method to find the value sought.

**boolean** isEmpty();

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

-simply check if the array is empty, return true if it is, false

If it’s not.

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

\* Removes the element at the specified index. Close up the gap

\* in the array by moving all elements forward one step.

\* Amend your “size” attribute to reflect the removal of the element.

\* **@param** index of element to be removed, starting from index 0

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

\* **@throws** IndexOutOfBoundsException

See code below. First check lower and upper bounds of array.

There is an edge case we discovered when array is full and we’re calling the first remove, which is the last item. In this case, we have to check if the index called is equal to the size of the array, if so, we can skip the second array copy statement.

**if**(index < 0 || index > size-1){

**throw** **new** IndexOutOfBoundsException();

} // END BRACKET for index out of bounds check

**else**

{

remove = get(index);

size --;

String [] tempIndexes = **new** String[size];

System.*arraycopy*(Indexes, 0, tempIndexes, 0, index);

**if**(size != index)

{

System.*arraycopy*(Indexes, index+1, tempIndexes, index, size-(index+1));

} // END BRACKET for check skip this copy if first item

Indexes = tempIndexes;

**return** remove;

}// END BRACKT FOR remove ELSE

**int** size();

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

Return size; // size is an attribute of this class.

Class **Stack** – implements IStack interface

**void** push(String s);

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

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

**Top = new Node(s, top);**

**void** pop();

\* Removes the top element from the stack

If(empty())

Throw new EmptyStackException

Else

retValue = top.value; // return value gets value of top value

top = top.next; // new top gets value of top’s next value

return retValue;

String peek();

\* Returns the top element without removing it.

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

If(empty())

Throw new EmptyStackException

Else

Return top.value;

**int** size();

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

Return size; // this is an attribute

**boolean** isEmpty();

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

Return top == null;

Class **Application** – implements IApplication

**public** IList readInputFile();

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

\* as an element in an IList

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

* **Open file for input**
* **Iterate through file**
* **File UTF read data from file**
* **Fill array with data from file**
* **Close file**
* **Return an IList object containing an array filled with data**

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

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

\* **@param** output

* **Open file for output**
* **Iterate through array in IList object**
* **Write data to file**
* **Close file**

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

{

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

\""+output+"\".");

}

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

\* @param input the input string

\* @param output the output string

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

\* Reverses the String parameter.

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

\* **@return** the reversed string

* **Use string builder**
* **Starting at last position in array iterate backwards**
* **Fill output string with string data in reverse order**

Class **NODE** – implements INode

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** Node<E> implements INode{

// PRIVATE FIELDS in this class.

private String value;

private Node next;

E getData();

\* Returns the data stored in this node.

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

Return this.value;

**void** setData(E data);

\* Setter for data for this node.

\* **@param** data New data

This.value = data;

INode<E> getNext();

\* Returns the node next to this node.

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

Return this.next;

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

Sets node received as the next node to this node.

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

This.next = next;