**Course**: Principals of Software Development – ENSF 409

**Lab 2**

**Instructor**: M. Moshirpour

**Student Name**: Mitchell Sawatzky

**Date Submitted**: Jan 29, 2016

Exercise B

TriangleAL.java

import java.util.ArrayList;

public class TriangleAL {

private ArrayList <ArrayList <Integer>> triangle;

private int size;

TriangleAL(int n) {

//allocate array and fill it

size = n;

triangle = new ArrayList <ArrayList <Integer>>(n+1);

for (int i = 0; i < n; i++) {

triangle.add(new ArrayList <Integer>(i+1));

for (int j = 0; j <= i; j++) {

if (j == 0 || j == i)

triangle.get(i).add(1);

else

triangle.get(i).add(triangle.get(i-1).get(j-1) + triangle.get(i-1).get(j));

}

}

}

public int size() {

return size;

}

public void printTriangle() {

//print triangle to stdout

for (int i = 0; i < size; i++)

for (int j = 0; j <= i; j++)

System.out.printf((j == 0 ? "" : " ") + "%d" + (j == i ? "\n" : ""), triangle.get(i).get(j));

}

public ArrayList <Integer> sumRows() {

//array with the sum of each row

ArrayList <Integer> sum = new ArrayList <Integer>(size);

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

int rowSum = 0;

for (int j = 0; j <= i; j++)

rowSum += triangle.get(i).get(j);

sum.add(rowSum);

}

return sum;

}

public ArrayList <Integer> sumCols() {

//array with the sum of each column

ArrayList <Integer> sum = new ArrayList <Integer>();

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

sum.add(0);

for (int j = 0; j <= i; j++)

sum.set(j, sum.get(j) + triangle.get(i).get(j));

}

return sum;

}

public static void main(String[] args) {

if (args.length < 1 ) {

System.err.println("ERROR: No integer argument.");

System.exit(1);

}

for (int i = 0; i < args[0].length(); i++) {

if (!Character.isDigit(args[0].charAt(i))) {

System.err.println("ERROR: Argument is not an integer.");

System.exit(1);

}

}

TriangleAL pt = new TriangleAL(Integer.parseInt(args[0]));

pt.printTriangle();

ArrayList <Integer> sum\_rows = pt.sumRows();

System.out.println("\nHere are the sum of rows:");

for(int i =0; i < pt.size(); i++)

System.out.println(sum\_rows.get(i));

ArrayList <Integer> sum\_cols = pt.sumCols();

System.out.println("\nHere are the sum of columns:");

for(int i =0; i < pt.size(); i++)

System.out.printf( "%-5d", sum\_cols.get(i));

System.out.println();

}

}

Terminal Output

Mitchell@ttys000 10:11 {0} [2]$ java TriangleAL 12

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

1 10 45 120 210 252 210 120 45 10 1

1 11 55 165 330 462 462 330 165 55 11 1

Here are the sum of rows:

1

2

4

8

16

32

64

128

256

512

1024

2048

Here are the sum of columns:

12 66 220 495 792 924 792 495 220 66 12 1

Exercise C

SimpleList.java

/\*\*

\* Provides data feilds and methods to create a Java data-type

\* resembling a linked list.

\* The overall purpose of this file is to demonstrate that C++ code

\* can be transformed into Java code to accomplish the same thing, and

\* also to provide a starting point for learning to use JavaDoc

\* comments.

\*

\* @author Mitchell Sawatzky

\* @version 1.0

\* @since January 21, 2016

\*/

public class SimpleList {

/\*\*

\* A helper method for the main method

\*/

private static void print(SimpleList list) {

for (int i = 0; i < list.size(); i++)

System.out.print(list.get(i) + " ");

}

/\*\*

\* Provides a simple structure to represent a single item (or node)

\* in a linked list.

\*/

private static class Node {

/\*\*

\* The integer value of a node

\*/

protected int item;

/\*\*

\* The reference to the next node in the list

\*/

protected Node next;

}

/\*\*

\* The first node in the linked list

\*/

private Node headM;

/\*\*

\* The number of nodes in the linked list

\*/

private int sizeM;

/\*\*

\* Constructs a SimpleList object with no nodes.

\*/

SimpleList() {

headM = null;

sizeM = 0;

}

/\*\*

\* Returns the amount of nodes in a SimpleList

\* @return Integer amount of nodes

\*/

public int size() {

return sizeM;

}

/\*\*

\* Adds a Node with an item to the end of the list and increments

\* sizeM.

\* @param item the Integer belonging to the new Node

\*/

public void push\_back(final int item) {

Node new\_node = new Node();

if (new\_node == null) {

System.out.println("\nNo memory available to create a node");

System.exit(1);

}

new\_node.item = item;

if (headM == null) {

new\_node.next = headM;

headM = new\_node;

} else {

Node p = headM;

while (p.next != null)

p = p.next;

p.next = new\_node;

new\_node.next = null;

}

sizeM++;

}

/\*\*

\* Adds a Node with an item to the beginning of the list and

\* increments sizeM.

\* @param item the Integer belonging to the new Node

\*/

public void push\_front(final int item) {

Node new\_node = new Node();

new\_node.item = item;

new\_node.next = headM;

headM = new\_node;

sizeM++;

}

/\*\*

\* Removes the last Node in the list and decrements sizeM.

\* @param item the item to remove...? idk man this function

\* isn't defined.

\*/

public void pop\_back(final int item) {

// Prototype defined in SimpleList.h, but the function

// definition is not included in SimpleList.h, SimpleList.cpp,

// or useSimpleList.cpp. Here is a java implementation regardless

Node p = headM;

while (p.next != null) {

if (p.next.next == null)

p.next = null;

else

p = p.next;

}

}

/\*\*

\* The item Integer at the nth position in the list is returned.

\* If n is less than 0 or greater than or equal to sizeM, the

\* program exits.

\* @param n the Integer location of the Node.

\* @return Integer contents of item at node n

\*/

public int get(int n) {

if (n < 0 || n >= sizeM) {

System.out.println("Illegal Access. Program Terminates...");

System.exit(1);

}

Node p = headM;

for (int i = 0; i < n; i++)

p = p.next;

return p.item;

}

/\*\*

\* Assigns the value of v to the item feild at Node n.

\* @param n index of node

\* @param v Integer value to set the item to.

\*/

public void set(int n, int v) {

if (n < 0 || n >= sizeM) {

System.out.println("Illegal Access. Program Terminates...");

System.exit(1);

}

Node p = headM;

for (int i = 0; i < n; i++)

p = p.next;

p.item = v;

}

/\*\*

\* A node with a copy of itemA is inserted into the nth position

\* of the list, and sizeM is incremented accordingly.

\* @param itemA the Integer value of the item at the new Node

\* @param n the Integer index of the Node to be inserted

\*/

public void insert(final int itemA, int n) {

if (n < 0 || n > sizeM)

return;

else if (n == 0)

this.push\_front(itemA);

else if (n == sizeM)

this.push\_back(itemA);

else {

Node new\_node = new Node();

if (new\_node == null) {

System.out.println("Sorry, memory is unavailable to create a new node.");

return;

}

new\_node.item = itemA;

Node before = headM;

Node after = headM.next;

int i = 1;

while (i < n) {

before = after;

after = after.next;

i++;

}

new\_node.next = after;

before.next = new\_node;

sizeM++;

}

}

/\*\*

\* Removes the Node in the nth position.

\* @param n the Integer index of the Node to remove

\*/

public void remove(int n) {

if (headM == null || n < 0 || n >= sizeM)

return;

Node be\_deleted;

Node before;

if (n == 0) {

be\_deleted = headM;

headM = headM.next;

} else {

before = headM;

be\_deleted = before.next;

int i = 1;

while (i < n) {

before = be\_deleted;

be\_deleted = before.next;

i++;

}

before.next = be\_deleted.next;

}

be\_deleted = null;

sizeM--;

}

/\*\*

\* Deletes all Nodes in the SimpleList.

\*/

public void clear() {

Node p = headM;

headM = null;

sizeM = 0;

}

public static void main(String[] args) {

SimpleList list = new SimpleList();

System.out.println("List just after creation -- is empty.");

list.push\_front(50);

System.out.println("After calling push\_front. list must have: 50");

print(list);

list.push\_back(440);

list.set(0,770);

System.out.println("\nAfter calling push\_back and set function list must have: 770 440");

print(list);

list.push\_back(330);

list.push\_back(220);

list.push\_back(110);

System.out.println("\nAfter three more calls to push\_back, list must have: 770, 440, 330, 220, 110");

print(list);

list.remove(0);

list.remove(2);

System.out.println("\nAfter removing two nodes. list must have: 440, 330, 110");

print(list);

list.insert(40, 3); //insert node with the value of 40 at the 4th position

list.insert(20, -1); // do nothing

list.insert(30, 30000); // do nothing

list.insert(10, 0); //insert node with the value of 10 at the 1st position

list.insert(33, 2); // insert node with the value 33 at the 3rd position

System.out.println("\nTwo more nodes inserted, must have: 10, 440, 33, 330, 110, 40");

print(list);

list.remove(0);

list.remove(1);

list.remove(2);

list.remove(3);

list.remove(4);

list.remove(5);

System.out.println("\nAfter 6 removes, list must have: 440, 330, 40: ");

print(list);

list.clear();

System.out.println("\nAfter call to clear, list must be empty:");

print(list);

list.push\_back(331);

list.push\_back(221);

list.push\_back(111);

System.out.println("\nAfter three calls to push\_back, list must have: 331, 221, 111");

print(list);

}

}

Terminal Output

Mitchell@ttys000 17:27 {0} [2]$ java SimpleList

List just after creation -- is empty.

After calling push\_front. list must have: 50

50

After calling push\_back and set function list must have: 770 440

770 440

After three more calls to push\_back, list must have: 770, 440, 330, 220, 110

770 440 330 220 110

After removing two nodes. list must have: 440, 330, 110

440 330 110

Two more nodes inserted, must have: 10, 440, 33, 330, 110, 40

10 440 33 330 110 40

After 6 removes, list must have: 440, 330, 40:

440 330 40

After call to clear, list must be empty:

After three calls to push\_back, list must have: 331, 221, 111

331 221 111