Harry Han

M. Moshipour

Jan 21, 2015 (Prelab)

Jan 29, 2015

Lab 2

ENSF 409 – Principles of Software Development

# Exercise A

Java Code:

/\*\*

\* Provides a Pascal's Triangle.

\*

\* The overall purpose is to learn how to use arrays.

\*

\* @author Harry Han

\* @version 1.0

\* @since January 22, 2016

\*/

public class Triangle {

/\*\*

\* Pascal Triangle

\*/

private int [][] triangle;

private int size;

public Triangle() {

/\*\*

\* The default constructor

\*/

}

public Triangle (int s) {

/\*\*

\* The overloaded constructor.

\* Generates the Triangle with to the assigned size.

\*/

size = s;

triangle = new int [size][];

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

triangle[i] = new int [i + 1];

}

this.fillTriangle();

}

public void fillTriangle() {

/\*\*

\* Fills the generated Triangle

\*/

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

for (int j = 0; j < i + 1; j++) {

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

triangle[i][j] = 1;

}

else {

triangle[i][j] = triangle[i-1][j-1] + triangle[i-1][j];

}

}

}

}

public void printTriangle() {

/\*\*

\* Prints the generated triangle

\*/

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

for (int j = 0; j < i + 1; j++) {

System.out.printf("%-5d", triangle[i][j]);

}

System.out.println();

}

}

public int[] sumRows() {

/\*\*

\* Sums all the rows of the generated triangle

\*/

int[] arr = new int[size];

int sum = 0;

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

for (int j = 0; j < i + 1; j++) {

sum+=triangle[i][j];

}

arr[i] = sum;

sum = 0;

}

return arr;

}

public int[] sumCols() {

/\*\*

\* Sums all the columns of the generated triangle

\*/

int[] arr = new int[size];

int sum = 0;

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

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

sum+=triangle[j+i][i];

}

arr[i] = sum;

sum = 0;

}

return arr;

}

public int size() {

/\*\*

\* Returns the size of the triangle (rows)

\*/

return size;

}

public static void main(String[] args) {

if(args.length < 1 ) {

System.err.println("Sorry this program needs an integer argument.");

System.exit(1);

}

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

pt.printTriangle();

int [] 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[i]);

int [] sum\_cols = pt.sumCols();

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

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

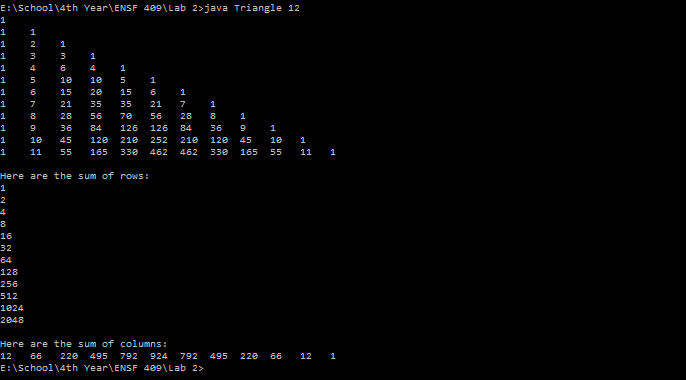
System.out.printf( "%-5d", sum\_cols[i]);

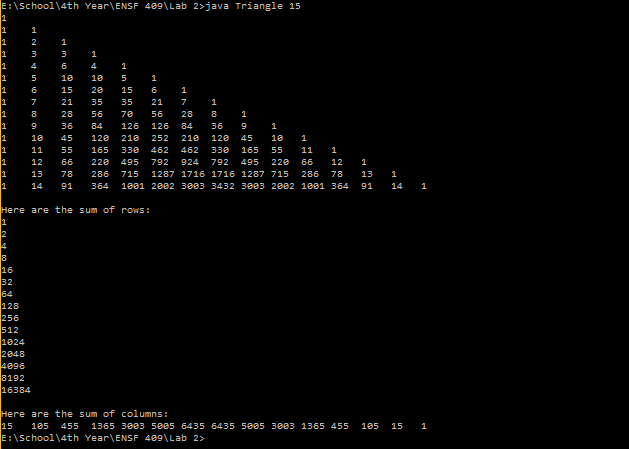
System.out.println();

}

}

Output:







# Exercise B

import java.util.ArrayList;

public class TriangleAL {

ArrayList<ArrayList<Integer>> triangle;

private int size;

public TriangleAL() {

}

public TriangleAL (int s) {

size = s;

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

this.fillTriangle();

}

public void fillTriangle() {

int row = 0;

while(row < size) {

triangle.add(new ArrayList<Integer>());

row++;

}

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

for (int j = 0; j < i + 1; j++) {

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

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

}

else {

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

}

}

}

}

public void printTriangle() {

/\*\*
\* Prints the generated triangle
\*/

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

for (int j = 0; j < i + 1; j++) {

System.out.printf("%-5d", triangle.get(i).get(j));

}

System.out.println();

}

}

public ArrayList<Integer> sumRows() {

/\*\*
\* Sums all the rows of the generated triangle
\*/

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

int sum = 0;

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

for (int j = 0; j < i + 1; j++) {

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

}

arr.add(i, sum);

sum = 0;

}

return arr;

}

Java Cod

public ArrayList<Integer> sumCols() {

/\*\*
\* Sums all the columns of the generated triangle
\*/

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

int sum = 0;

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

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

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

}

arr.add(i, sum);

sum = 0;

}

return arr;

}

public int size() {

/\*\*
\* Returns the size of the triangle (rows)
\*/

return size;

}

public static void main(String[] args) {

if(args.length < 1 ) {

System.err.println("Sorry this program needs an integer argument.");

System.exit(1);

}

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

pt.printTriangle();

ArrayList arr = new ArrayList();

arr = pt.sumRows();

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

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

System.out.println(arr.get(i));

arr.clear();

arr = pt.sumCols();

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

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

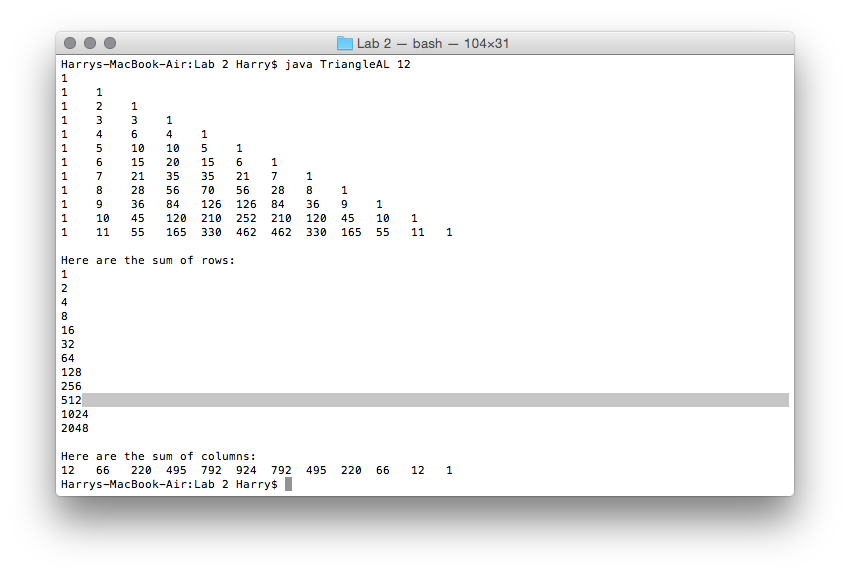
System.out.printf( "%-5d", arr.get(i));

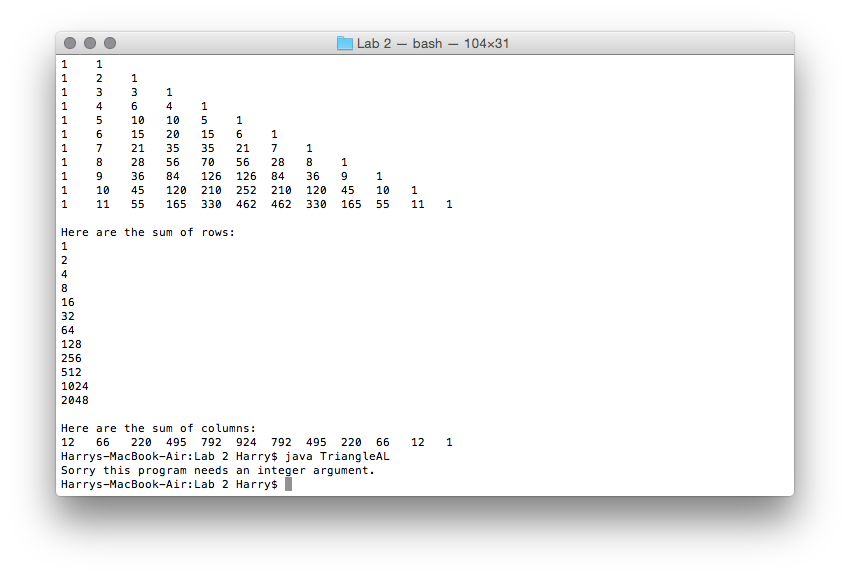
System.out.println();

}

}

Output:





# Exercise C

Java Code:

/\*\*

\* Translating a simple linked list from C++ to Java

\* The overall purpose of the lab is to translate the differences

\* between C++ and Java in terms of linked lists. Although there is

\* a LinkedList object that already exists in Java, this code replicates

\* the object in terms of C++.

\*

\* @author Harry Han

\* @version 1.0

\* @since January 27, 2016

\*/

public class SimpleList {

/\*\*

\* A node class which contains the data and the next node address

\*/

private Node head;

/\*\*

\* The total size of the linked list

\*/

private int sizeM;

/\*\*

\* Constructs a LinkedList object without any specified values. Automatically

\* sets the size to be 0 and the head to be a null node

\*/

public SimpleList() {

head = new Node(null);

sizeM = 0;

}

/\*\*

\* Gets the data from the specified node location

\*

\* @param n the location of the node

\*/

public int get(int n) {

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

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

System.exit(1);

}

Node p = head;

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

p = p.getNext();

}

return p.getData();

}

/\*\*

\* Sets the data stored inside a specified Node

\*

\* @param n the location of the node

\* @param v the data to be stored in the node

\*/

public void set(int n, int v){

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

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

System.exit(1);

}

Node p = head;

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

p = p.getNext();

}

p.setData(v);

}

/\*\*

\* Adds a new node to the end of the linked list. If the linked list is empty

\* then the program will add the node to the head.

\*

\* @param item the data to be stored in the node

\*/

public void push\_back(int item) {

Node new\_node = new Node(null);

if (new\_node == null) {

System.out.println("No memory available to create a node");

System.exit(1);

}

new\_node.setData(item);

Node p = head;

if(p.getData() == null) {

new\_node.setNext(null);

p = new\_node;

head = p;

}

else {

while (p.getNext() != null) {

p = p.getNext();

}

p.setNext(new\_node);

new\_node.setNext(null);

}

sizeM++;

}

/\*\*

\* Adds a new node to the start of the linked list and assigns the head to the

\* new node

\*

\* @param item the data to be stored in the node

\*/

public void push\_front(int item) {

Node new\_node = new Node(null);

new\_node.setData(item);

if(sizeM == 0) {

new\_node.setNext(null);

}

else {

new\_node.setNext(head);

}

head = new\_node;

sizeM++;

}

/\*\*

\* Inserts a node at a specified location in the linked list. If the location

\* specified is at the start of the linked list, then the function will call

\* push\_front. If the location is at the end, the push\_back will be called.

\*

\* @param itemA the data to be stored in the node

\* @param n the location of the node

\*/

public void insert(int itemA, int n) {

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

return;

}

else if(n == 0) {

push\_front(itemA);

}

else if(n == sizeM) {

push\_back(itemA);

}

else {

Node new\_node = new Node(null);

if (new\_node == null) {

System.out.println("No memory available to create a node");

System.exit(1);

}

new\_node.setData(itemA);

Node before = head;

Node after = head.getNext();

int i = 1;

while (i < n) {

before = after;

after = after.getNext();

i++;

}

new\_node.setNext(after);

before.setNext(new\_node);

sizeM++;

}

}

/\*\*

\* Completely deletes the entire Linked List. This is done by setting each node to

\* null

\*/

public void clear() {

Node p = head;

while(p != null) {

head = head.getNext();

p = null;

p = head;

}

head = new Node(null);

sizeM = 0;

}

/\*\*

\* Removes a node from the specified location

\*

\* @param n the location of the node to be removed

\*/

public void remove(int n) {

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

return;

}

Node before = new Node(null);

Node be\_deleted = new Node(null);

if(n == 0) {

be\_deleted = head;

head = head.getNext();

}

else {

before = head;

be\_deleted = before.getNext();

int i = 1;

while (i < n) {

before = be\_deleted;

be\_deleted = before.getNext();

i++;

}

before.setNext(be\_deleted.getNext());

}

sizeM--;

}

/\*\*

\* Returns the size of the linked list

\*/

public int size() {

return sizeM;

}

private class Node {

/\*\*

\* The address of the next node

\*/

Node next;

/\*\*

\* The data stored in the linked list

\*/

Integer item;

/\*\*

\* Default constructor

\*/

public Node() {

}

/\*\*

\* Constructs a node object with the specified data, without knowledge

\* of the next node.

\*

\* @param data the data to be stored

\*/

public Node(Integer data) {

next = null;

item = data;

}

/\*\*

\* Constructs a node object with the specified data and next node

\*

\* @param \_next the address of the next node

\* @param data the data to be stored

\*/

public Node(Node \_next, Integer data){

next = \_next;

item = data;

}

public Integer getData() {

return item;

}

public void setData(Integer data) {

item = data;

}

public Node getNext() {

return next;

}

public void setNext(Node \_next) {

next = \_next;

}

}

/\*\*

\* Prints the linked list

\*

\* @param list the linked list to be printed

\*/

public static void print(SimpleList list) {

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

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

}

}

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);

System.out.println();

}

}

Output:

