CircularDoubleLinkedList Class:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom02;

/\*\*

\* Circular version of double linked list discussed in Chapter 2.

\* @author Jacob Huesman

\*/

public class CircularDoubleLinkedList<E> implements Cloneable {

//-----------nested Node class ------------------

private static class Node<E> {

private E element;

private Node<E> prev;

private Node<E> next;

public Node(E e, Node<E> p, Node<E> n) {

element = e;

prev = p;

next = n;

}

public E getElement(){ return element; }

public Node<E> getPrev() { return prev; }

public Node<E> getNext() { return next; }

public void setPrev(Node<E> p) { prev = p; }

public void setNext(Node<E> n) { next = n; }

}

//----------end of nested Node class-------------

//instance variables of the CircularDoubleLinkedList

private Node<E> trailer;

private int size;

/\*\*

\* Constructs a new empty list.

\*/

public CircularDoubleLinkedList(){

trailer = null;

size = 0;

}

/\*\*

\* Returns the number of elements in the linked list.

\* @return int

\*/

public int size() {

return size;

}

/\*\*

\* Returns if the List is empty.

\* @return true - if empty, false - otherwise

\*/

public boolean isEmpty() {

return size == 0;

}

/\*\*

\* Returns the current first element in the list.

\* @return E

\*/

public E first() {

if(isEmpty()) {

return null;

}

return trailer.getNext().getElement();

}

/\*\*

\* Returns the current last element in the list.

\* @return E

\*/

public E last() {

if(isEmpty()) {

return null;

}

return trailer.getElement();

}

//public update methods

/\*\*

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

\* @param e element to be added

\*/

public void addFirst(E e) {

if(isEmpty()) {

trailer = new Node<E>(e,null,null);

trailer.setNext(trailer);

trailer.setPrev(trailer);

} else {

addBetween(e, trailer, trailer.getNext());

}

size++;

}

/\*\*

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

\* @param e element to be added

\*/

public void addLast(E e) {

addFirst(e);

trailer = trailer.getNext();

}

/\*\*

\* Removes and returns the first element of the list.

\* @return first element

\*/

public E removeFirst() {

if(isEmpty()) {

return null;

}

return remove(trailer.getNext());

}

/\*\*

\* Removes and returns the last element of the list.

\* @return last element

\*/

public E removeLast() {

if(isEmpty()) {

return null;

}

trailer = trailer.getPrev();

return remove(trailer.getNext());

}

/\*\*

\* Rotate list forwards one

\*/

public void rotate() {

trailer = trailer.getNext();

}

/\*\*

\* Rotates list backwards one

\*/

public void rotateBackward() {

trailer = trailer.getPrev();

}

/\*\*

\* Implements shallow cloning

\* @return shallow clone of the object

\* @throws CloneNotSupportedException

\*/

@Override

public Object clone() throws CloneNotSupportedException {

return super.clone();

}

/\*\*

\* Checks to see if the elements of the list passed as an argument match the elements of the current list.

\* @param list2 list to check against

\* @return true - if they are equal; false otherwise

\*/

public boolean equals(CircularDoubleLinkedList list2) {

if(this.size() != list2.size()) {

return false;

} else {

try {

//Don't want to mess up current order of list2.

CircularDoubleLinkedList list = (CircularDoubleLinkedList) list2.clone();

Node<E> current = trailer;

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

//Check to see if current element corresponds to the current last element of list.

//If yes continue checking the rest of the elements.

//If no proceed to the element after current.

if(current.getElement().equals(list.last())){

Node<E> current2 = current;

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

list.rotate();

current2 = current2.getNext();

if(!current2.getElement().equals(list.last())){

list = (CircularDoubleLinkedList) list2.clone();

break;

} else if(a == size-1) {

return true;

}

}

}

current = current.getNext();

}

return false;

} catch(CloneNotSupportedException e){

return false;

}

}

}

//private update methods

/\*\*

\* Adds element e to the linked list in between the given nodes.

\* @param e element to be added

\* @param predecessor previous node

\* @param successor next node

\*/

private void addBetween(E e, Node<E> predecessor, Node<E> successor) {

// create and link a new node

Node<E> newest = new Node<>(e, predecessor, successor);

predecessor.setNext(newest);

successor.setPrev(newest);

}

/\*\*

\* Removes the given node from the list and returns its element.

\* @param node node to be removed

\* @return node that was removed

\*/

private E remove(Node<E> node) {

Node<E> predecessor = node.getPrev();

Node<E> successor = node.getNext();

predecessor.setNext(successor);

successor.setPrev(predecessor);

size--;

return node.getElement();

}

}

Test Program:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom02;

import java.util.Random;

/\*\*

\* Test class to demonstrate the various methods of the Circularly Linked List

\* @author Jacob Huesman

\*/

public class Test {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

CircularDoubleLinkedList<Integer> list = new CircularDoubleLinkedList<>();

Random random = new Random();

System.out.println("Creating a CircularDoubleLinkedList of 10 random integers between 10-50...");

for(int i=0; i<10; i++){

list.addLast(random.nextInt(40)+11);

}

//demonstrate size()

System.out.println("Current size: " + list.size());

//demonstrate isEmpty()

System.out.println("Empty?: " + list.isEmpty());

System.out.println("Creating an empty list...");

CircularDoubleLinkedList<Integer> empty = new CircularDoubleLinkedList<>();

System.out.println("Empty?: " + empty.isEmpty());

System.out.println("Back to the original list...");

//demonstrate first()

System.out.println("The first element is: " + list.first());

//demonstrate last()

System.out.println("The last element is: " + list.last());

//demonstrate addFirst()

System.out.println("Adding the number 98 to the front of the list...");

list.addFirst(98);

System.out.println("The first element is: " + list.first());

//demonstrate addLast()

System.out.println("Addding the number 56 to the back of the list...");

list.addLast(56);

System.out.println("The last element is: " + list.last());

//demonstrate removeFirst()

System.out.println("Removing " + list.removeFirst() + " from the front of the list...");

System.out.println("The new first element is: " + list.first());

//demonstrate removeLast()

System.out.println("Removing " + list.removeLast() + " from the back of the list...");

System.out.println("The new last element is: " + list.last());

//demonstrate rotate()

System.out.println("After rotating once the first element should be the last element.");

System.out.println("Current first element: " + list.first());

System.out.println("Rotating...");

list.rotate();

System.out.println("New last element: " + list.last());

//demonstrate rotateBackward()

System.out.println("After rotating once backwards the last element should be the first element.");

System.out.println("Current last element: " + list.last());

System.out.println("Rotating backwards...");

list.rotateBackward();

System.out.println("New first element: " + list.first());

//demonstrate clone()

System.out.println("After a shallow clone the elements of the cloned list should point to the same elements of the original array.");

try {

CircularDoubleLinkedList copy = (CircularDoubleLinkedList<Integer>) list.clone();

String[][] array = new String[list.size()+1][2];

array[0][0] = "Original";

array[0][1] = "Cloned";

for(int i=1; i<list.size()+1; i++){

array[i][0] = list.first().toString();

list.rotate();

array[i][1] = copy.first().toString();

copy.rotate();

}

System.out.println(ConsoleTable.makeTable("Element Comparison", 50, array));

//demonstrate equals()

System.out.println("Using the equals method the original list should equal the cloned list.");

if(list.equals(copy)){

System.out.println("They are equal!");

} else {

System.out.println("They are not equal.");

}

System.out.println("");

//demonstrate equals method on rotated copy

System.out.println("Note: The equals method will work even when the lists are rotated.");

copy.rotateBackward();

array = new String[list.size()+1][2];

array[0][0] = "Original";

array[0][1] = "Cloned";

for(int i=1; i<list.size()+1; i++){

array[i][0] = list.first().toString();

list.rotate();

array[i][1] = copy.first().toString();

copy.rotate();

}

System.out.println(ConsoleTable.makeTable("Element Comparison", 50, array));

//demonstrate equals()

copy.rotateBackward();

System.out.println("Using the equals method the original array should equal the cloned array.");

if(list.equals(copy)){

System.out.println("They are equal!");

} else {

System.out.println("They are not equal.");

}

} catch (Exception e) {

System.out.println("Clone operation failed!");

}

}

}

Test Program Output:

run:

Creating a CircularDoubleLinkedList of 10 random integers between 10-50...

Current size: 10

Empty?: false

Creating an empty list...

Empty?: true

Back to the original list...

The first element is: 34

The last element is: 44

Adding the number 98 to the front of the list...

The first element is: 98

Addding the number 56 to the back of the list...

The last element is: 56

Removing 98 from the front of the list...

The new first element is: 34

Removing 56 from the back of the list...

The new last element is: 44

After rotating once the first element should be the last element.

Current first element: 34

Rotating...

New last element: 34

After rotating once backwards the last element should be the first element.

Current last element: 34

Rotating backwards...

New first element: 34

After a shallow clone the elements of the cloned list should point to the same elements of the original array.

+-----------------------------------------------+

| Element Comparison |

+-----------------------+-----------------------+

| Original | Cloned |

+-----------------------+-----------------------+

| 34 | 34 |

+-----------------------+-----------------------+

| 34 | 34 |

+-----------------------+-----------------------+

| 48 | 48 |

+-----------------------+-----------------------+

| 24 | 24 |

+-----------------------+-----------------------+

| 26 | 26 |

+-----------------------+-----------------------+

| 13 | 13 |

+-----------------------+-----------------------+

| 27 | 27 |

+-----------------------+-----------------------+

| 26 | 26 |

+-----------------------+-----------------------+

| 25 | 25 |

+-----------------------+-----------------------+

| 44 | 44 |

+-----------------------+-----------------------+

Using the equals method the original list should equal the cloned list.

They are equal!

Note: The equals method will work even when the lists are rotated.

+-----------------------------------------------+

| Element Comparison |

+-----------------------+-----------------------+

| Original | Cloned |

+-----------------------+-----------------------+

| 34 | 44 |

+-----------------------+-----------------------+

| 34 | 34 |

+-----------------------+-----------------------+

| 48 | 34 |

+-----------------------+-----------------------+

| 24 | 48 |

+-----------------------+-----------------------+

| 26 | 24 |

+-----------------------+-----------------------+

| 13 | 26 |

+-----------------------+-----------------------+

| 27 | 13 |

+-----------------------+-----------------------+

| 26 | 27 |

+-----------------------+-----------------------+

| 25 | 26 |

+-----------------------+-----------------------+

| 44 | 25 |

+-----------------------+-----------------------+

Using the equals method the original array should equal the cloned array.

They are equal!

BUILD SUCCESSFUL (total time: 0 seconds)

Pseudo-Code:

Simple Bubble Sort for integers

**Algorithm:** bubbleSort( array ) :

**Input:** integer array named to be sorted

**Output:** sorted integer array

boolean swap = true { Swap is used to keep track of if any elements were swapped in the most recent loop. If none were then the array is sorted and can be returned. }

**for** i=0 to i<array.length-1 **do**

**if** swap != true **then**

**break**

**else**

swap = false

make first element in array the current element

**for** j=1 to j<=array.length **do**

**if** current element > element j **then**

swap = true;

swap the elements

**return** array

Answers to Task 2:

Big-Oh of simple bubble sort:

The bubble sort contains two loops, one nested in the other. This indicates that if the algorithm were to execute every loop fully, the worst case scenario, the Big O notation would be O(n2). Since it would be executing the code contained in the loops n2 times. The best case scenario would have a Big O notation of O(n). In the best case the array would be passed already sorted and after one loop the if statement would break out of the loop and the array would be returned.

Experiemental Big-Oh for simple bubble sort:

Unfortunately my computer runs the sort too fast for me to be able to determine the Big-Oh based on array sizes less than or equal to 100,000. I had to up it to 1,000,000 before the sort started to take more than a few milliseconds. Up to a million the average times were as follows 0, 0, 3, 3, 12. These results are not linear, jumping from 0 to 3 to 12. They could fall under a quadratic curve though if scaled with a constant c. The constant in this case would be 3\*10-8. Since the data falls under the curve c\*n2 for an n value greater than 1, we can say that the function has a Big-Oh of O(n2).

Determination of length of time to sort 1,000,000 random integers. If I use the constant from the previous example the answer would be 30,000. Which is pretty far off from the experiemental value of 12. However 12 is underneath the curve, so it is a valid.

SimpleBubbleSort class:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom02;

import java.util.ArrayList;

import java.util.Random;

/\*\*

\* Static class that implements and tests a simple bubble sort techniques.

\* @author Jacob Huesman

\*/

public class SimpleBubbleSort {

//Static Class

private SimpleBubbleSort(){}

/\*\*

\* Creates an int array containing all the test data.

\* @return the int array

\*/

public static int[][] bubbleSortTest() {

Random rand = new Random();

int[][] stats = new int[20][2];

for( int i=100, j=0; i<=1000000; i = i\*10, j++ ){

for( int k=0; k<3; k++){

stats[j\*4+k][0] = i;

ArrayList<Integer> list = new ArrayList<>(i);

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

list.add(rand.nextInt());

}

long start = System.currentTimeMillis();

sort(list);

long end = System.currentTimeMillis();

long dur = end - start;

stats[j\*4+k][1] = (int) dur;

if( k==2 ){

stats[j\*4+3][0] = i;

stats[j\*4+3][1] = (( stats[j\*4+2][1] + stats[j\*4+1][1] + stats[j\*4][1]) / 3);

}

}

}

return stats;

}

/\*\*

\* Sorts an array list using simple bubble sort technique

\* @param array array to be sorted

\* @return the sorted array

\*/

public static ArrayList<Integer> sort( ArrayList<Integer> array ) {

boolean swap = true;

for( int i=0; i<array.size()-1; i++ )

if(!swap){

break;

} else {

swap = false;

for( int j=1, temp=0; j<array.size(); j++ ){

if( array.get(j-1) > array.get(j) ){

swap = true;

temp = array.get(j-1);

array.set(j, array.get(j-1));

array.set(j-1, temp);

}

}

}

return array;

}

}

Client:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom02;

import java.util.ArrayList;

import java.util.Random;

/\*\*

\* Static class that implements and tests a simple bubble sort techniques.

\* @author Jacob Huesman

\*/

public class SimpleBubbleSort {

//Static Class

private SimpleBubbleSort(){}

/\*\*

\* Creates an int array containing all the test data.

\* @return the int array

\*/

public static int[][] bubbleSortTest() {

Random rand = new Random();

int[][] stats = new int[20][2];

for( int i=100, j=0; i<=1000000; i = i\*10, j++ ){

for( int k=0; k<3; k++){

stats[j\*4+k][0] = i;

ArrayList<Integer> list = new ArrayList<>(i);

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

list.add(rand.nextInt());

}

long start = System.currentTimeMillis();

sort(list);

long end = System.currentTimeMillis();

long dur = end - start;

stats[j\*4+k][1] = (int) dur;

if( k==2 ){

stats[j\*4+3][0] = i;

stats[j\*4+3][1] = (( stats[j\*4+2][1] + stats[j\*4+1][1] + stats[j\*4][1]) / 3);

}

}

}

return stats;

}

/\*\*

\* Sorts an array list using simple bubble sort technique

\* @param array array to be sorted

\* @return the sorted array

\*/

public static ArrayList<Integer> sort( ArrayList<Integer> array ) {

boolean swap = true;

for( int i=0; i<array.size()-1; i++ )

if(!swap){

break;

} else {

swap = false;

for( int j=1, temp=0; j<array.size(); j++ ){

if( array.get(j-1) > array.get(j) ){

swap = true;

temp = array.get(j-1);

array.set(j, array.get(j-1));

array.set(j-1, temp);

}

}

}

return array;

}

}

ConsoleTable class used to display table:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom02;

import java.util.Formatter;

import java.util.Locale;

/\*\*

\* Creates a basic table that can be output to the console.

\* @author Jacob Huesman

\*/

public class ConsoleTable {

//Static class

private ConsoleTable(){}

/\*\*

\* Makes a table with a title.

\* @param title String that will be displayed as the title

\* @param width The preferred width of the table output. May be slightly modified to ensure columns are spaced evenly.

\* @param array Two dimensional String array that holds the data to be displayed in the table

\* @return A formatted string that when output to the console will be a table of data

\*/

public static String makeTable(String title, int width, String[] columnNames, String[][] array) {

StringBuilder sb = new StringBuilder();

Formatter format = new Formatter(sb, Locale.US);

int columns = array[0].length;

//Practical width

int wid = ((width-1)/columns)\*columns+1;

//Create a centered title

sb.append(addSeperator(wid));

sb.append("|" + center(title, wid-2) + "|\n");

sb.append(addSeperator(wid));

String[][] col = new String[1][];

col[0] = columnNames;

//Add column names

sb.append(genTable(width, col));

//Create the rest of the table

return sb.append(genTable(width, array)).toString();

}

/\*\*

\* Makes a table with a title.

\* @param title String that will be displayed as the title

\* @param width The preferred width of the table output. May be slightly modified to ensure columns are spaced evenly.

\* @param array Two dimensional String array that holds the data to be displayed in the table

\* @return A formatted string that when output to the console will be a table of data

\*/

public static String makeTable(String title, int width, String[][] array){

StringBuilder sb = new StringBuilder();

int columns = array[0].length;

//Practical width

int wid = ((width-1)/columns)\*columns+1;

//Create a centered title

sb.append(addSeperator(wid));

sb.append("|" + center(title, wid-2) + "|\n");

//Create the rest of the table

return sb.append(makeTable(width, array)).toString();

}

private static String makeTable(int width, String[][] array){

StringBuilder sbY = new StringBuilder();

//Figure out rows and columns.

int rows = array.length;

int columns = array[0].length;

//Practical width

int wid = ((width-1)/columns)\*columns+1;

//Initial seperator

sbY.append(addSeparator(wid, columns));

return sbY.append(genTable(width, array)).toString();

}

/\*\*

\* Generates a table to be used by the other methods of the class.

\* @param width The preferred width of the table output. May be slightly modified to ensure columns are spaced evenly.

\* @param array Two dimensional String array that holds the data to be displayed in the table

\* @return A formatted string that when output to the console will be a table of data

\*/

private static String genTable(int width, String[][] array){

StringBuilder sbY = new StringBuilder();

StringBuilder sbX;

//Figure out rows and columns.

int rows = array.length;

int columns = array[0].length;

//Practical width

int wid = ((width-1)/columns)\*columns+1;

//Create rest of table

try {

for(int y=0; y<rows; y++){

sbX = new StringBuilder();

sbX.append("|");

for(int x=0; x<columns; x++){

sbX.append(center(array[y][x], (width-1)/columns-1) + "|");

}

sbY.append(sbX.toString() + "\n");

sbY.append(addSeparator(wid, columns));

}

} catch (ArrayIndexOutOfBoundsException e) {

throw new ArrayIndexOutOfBoundsException("There is at least one row in the array that does not have " + columns + " columns.");

}

return sbY.toString();

}

/\*\*

\* Used to center a string, given a desired length of an output string.

\* @param text The text to be centered

\* @param len The length of the output string

\* @return A string of length len with text centered

\*/

private static String center( String text, int len ) {

String out = String.format("%"+len+"s%s%"+len+"s", "", text, "");

float mid = (out.length()/2);

float start = mid - (len/2);

float end = start + len;

return out.substring((int)start, (int)end);

}

/\*\*

\* Creates a separator to separate the rows of the table, taking into account the columns of the table.

\* @param width Width of the separator

\* @param columns Columns to be represented

\* @return String to be used as a separator

\*/

private static char[] addSeparator(int width, int columns){

char[] sep = addSeperator(width);

for(int i=1; i<columns; i++){

sep[i\*(width-1)/columns] = '+';

}

return sep;

}

/\*\*

\* Creates a separator to separate the rows of the table.

\* @param width Width of the separator

\* @return String to be used as a separator

\*/

private static char[] addSeperator(int width){

char[] sep = new char[width+1];

sep[0] = '+';

sep[width-1] = '+';

sep[width] = '\n';

for(int i=1; i<width-1; i++){

sep[i] = '-';

}

return sep;

}

}

Client Output:

run:

+-----------------------------------------------------------------------------+

| Simple Bubble Sort |

+-----------------------------------------------------------------------------+

| Run | Length | Time |

+-------------------------+-------------------------+-------------------------+

| 1 | 100 | 0 |

+-------------------------+-------------------------+-------------------------+

| 2 | 100 | 0 |

+-------------------------+-------------------------+-------------------------+

| 3 | 100 | 0 |

+-------------------------+-------------------------+-------------------------+

| AVG | 100 | 0 |

+-------------------------+-------------------------+-------------------------+

| 1 | 1000 | 0 |

+-------------------------+-------------------------+-------------------------+

| 2 | 1000 | 0 |

+-------------------------+-------------------------+-------------------------+

| 3 | 1000 | 1 |

+-------------------------+-------------------------+-------------------------+

| AVG | 1000 | 0 |

+-------------------------+-------------------------+-------------------------+

| 1 | 10000 | 4 |

+-------------------------+-------------------------+-------------------------+

| 2 | 10000 | 4 |

+-------------------------+-------------------------+-------------------------+

| 3 | 10000 | 3 |

+-------------------------+-------------------------+-------------------------+

| AVG | 10000 | 3 |

+-------------------------+-------------------------+-------------------------+

| 1 | 100000 | 10 |

+-------------------------+-------------------------+-------------------------+

| 2 | 100000 | 2 |

+-------------------------+-------------------------+-------------------------+

| 3 | 100000 | 2 |

+-------------------------+-------------------------+-------------------------+

| AVG | 100000 | 4 |

+-------------------------+-------------------------+-------------------------+

| 1 | 1000000 | 24 |

+-------------------------+-------------------------+-------------------------+

| 2 | 1000000 | 12 |

+-------------------------+-------------------------+-------------------------+

| 3 | 1000000 | 12 |

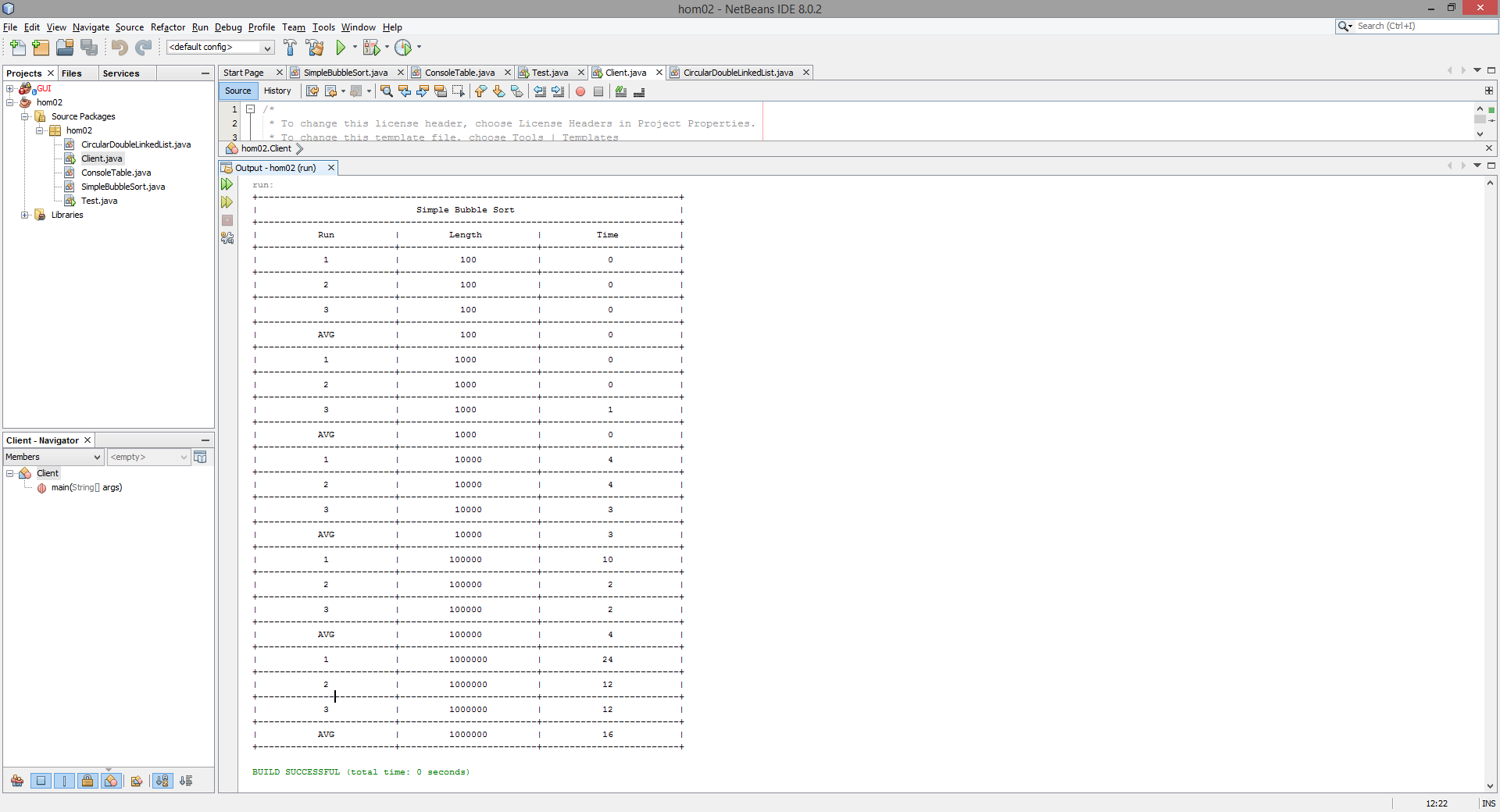
+-------------------------+-------------------------+-------------------------+

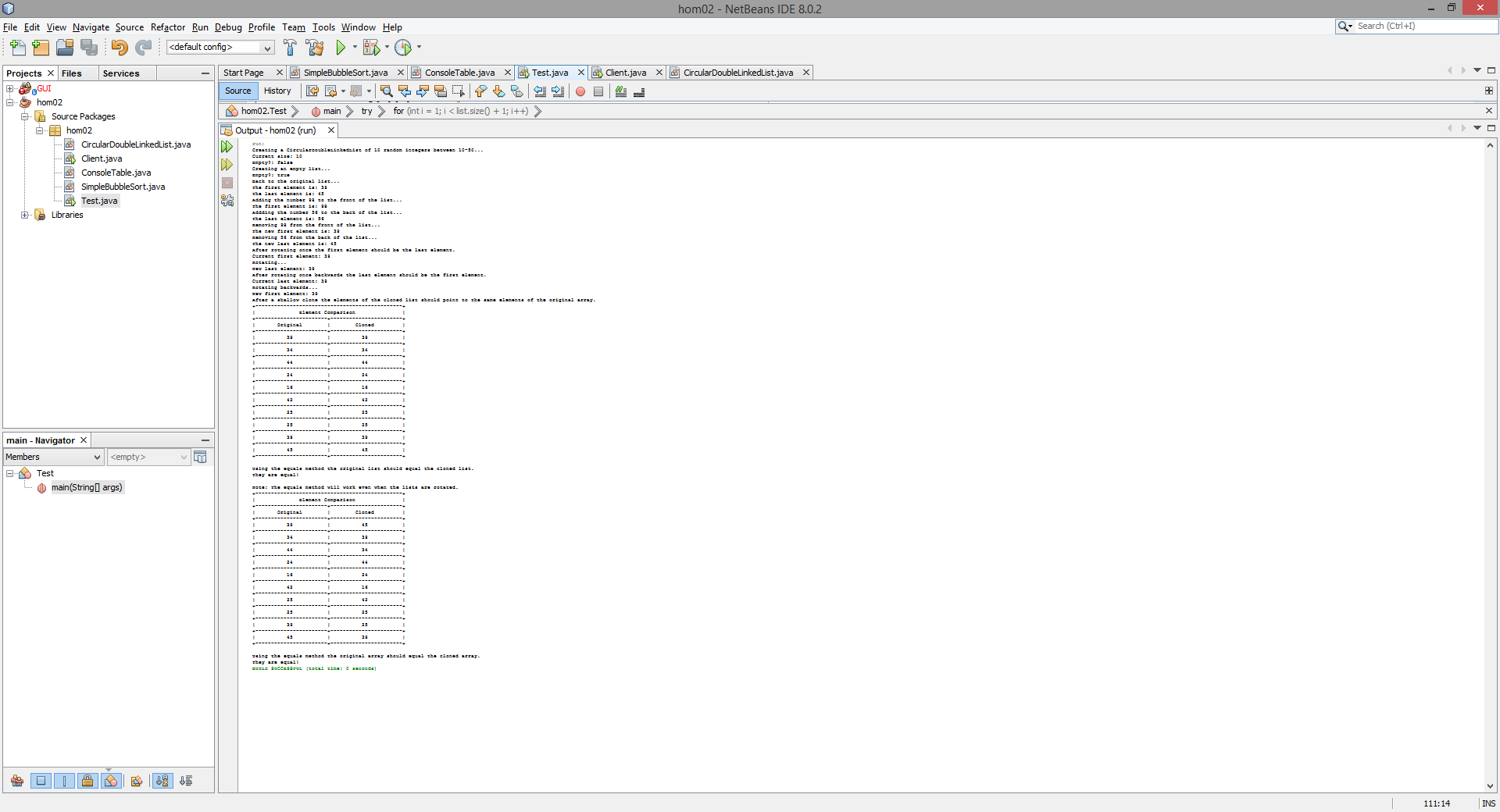
| AVG | 1000000 | 16 |

+-------------------------+-------------------------+-------------------------+

BUILD SUCCESSFUL (total time: 0 seconds)

Screenshots:





TASK 3:

C-3.28: Describe in detail an algorithm for reversing a SLL using only a constant amount of additional space.

Pseudo-Code:

**Algorithm:** reverseSLL ( SLL )

**Input:** A Singly Linked List to be reversed

**Output:** A reversed version of the SLL passed as a parameter

**Code:**

**if** head.next() != null **then**

currentReference = head.next()

prevReference = head

prevReference.next = null

tail = head

**else**

**return** SLL

**while** currentReference.next() != null **do**

prevReference = currentReference

currentReference = prevReference.next()

next = currentReference.next()

currentReference.next = prevReference

currentReference.next = prevReference

head = currentReference

**return** SLL

R-4.2. n0 is greater than 1

R-4.8.

- 210 is O(1)

- 2log(n) is O(n)

- 3n+100log(n) is O(n)

- 4n is O(n)

- 4nlog(n)+2n is O(nlog(n))

- nlog(n) is O(nlog(n))

- n2 + 10n is O(n2)

- n3 is O(n3)

- 2n is O(an)

R-4.9. O(n)

R-4.10. O(n)

R-4.11. O(n2)

R-4.12. O(n)

R-4.13. O(n3)