# Title:

# Mandatory Assignment 2 – LinkedList

### Team 5:

### Lukas Janocko

### Aimo Suikkanen

### James Testmann

### Sudhir Charusyia

### Shicheng Dai

### Supervisor:

### Rodger Munck Fairwood

## Perspective

Understand the internal structure of a linked‐based data structure and the efficiency of a range of

operations on the structure.

Applications may use your list library with a variety of algorithms. Such algorithms may expect functionality like:

* Insert and delete operations anywhere in the list
* Traversal of the list (several varieties possible)
* The list may be used to implement a stack and a queue (e.g. for depth first and breadth first search in trees). Queues and stacks are adaptor containers, i.e. they are based on another container. (For efficiency purposes it may be advisable to make a specific implementation for the stack and the queue, but here we assume your own linked list would be used.)
* Different kinds of sorting algorithms may use the linked list

## Assignment

Design and implement your own linked list.

After some analysis, some basic points must be considered in the design process:

* Single or double linked structure? Why?

The single linked structure contains two fields, one field of data and one field containing the address of the next node. The single linked structure might be useful for smaller lists, but the design requires a search to go through the list from beginning every time the index of a node is smaller than the previous.

The double linked structure contains one more field than the single linked structure. This added field contains the address of the previous node in the list.

This requires more memory than the single linked list. If the resources are large enough, the design of the linked list allows for traversal searching. Traversal searching means it is possible to go to the previous node in the search of a specific node. If the list size, the start node and the end node is known, it allows to choose from where to start a search accordingly.

According to the size of the used List, it has been chosen to use a double linked list.

* Circular or non‐circular structure? Why?

The circular structure does not have a null pointer at the end of the list (both ends if double linked). Allowing it to run through the list continuously. This disables the opportunity to add nodes to the start or end of the list.

For the assignment it is chosen not to use circular linked lists. The reason is that an end node and a start node is needed for the assignment.

* Sentinels or no sentinels? Why?

A sentinel node is used to make traversal searches easier. A sentinel node contains no data and is pointed at by the end node and the start node. This is used as an alternative to a null pointer check, giving the opportunity to continue a search, traversal or normal, eliminating steps and saving time in the search.

A sentinel comes at the cost of extra pointers and resource usage.

Given the cost versus the efficiency of a sentinel it is decided to use sentinels in the assignment.

## Results:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SORTING TESTS:** |  |  |  |  |  |  |  |  |  |  |  |  |
| **MyLinkedList** | | N1 = 500 elements | N3 = 1000 elements | N4 = 2000 elements | N5 = 4000 elements | **Timerelation for N2 / N1** | O(n^2) + O(n^2) O-notation for N2 / N1 | **Timerelation for N3 / N2** | O(n^2) + O(n^2) O-notation for N3 / N2 | **Timerelation for N4 / N3** | O(n^2) + O(n^2) O-notation for N4 / N3 |  |
| T(N1) run 20 times | T(N2) run 10 times | T(N3) run 5 times | T(N4) run 2 times |  |
| Random Collection | Selection Sort (O(n2)) | 102.24ms | 818.09ms | 6920.48ms | 59902.00ms | **8.00** | 8.00 | **8.46** | 8.00 | **8.66** | 8.00 | Expected difference because sorting linked lists is O(n^2) for many of the operations and another O(n^2) for the sorting! hence TWICE the time to sort. variance in insertion sort efficiency for nearly sorted collections is O(n^2) + O(n) |
| Insertion Sort (O(n2)) | 58.43ms | 460.82ms | 3829.88ms | 34174.00ms | **7.89** | 8.00 | **8.31** | 8.00 | **8.92** | 8.00 |
| Nearly Sorted Collection | Selection Sort (O(n2)) | 87.22ms | 696.77ms | 5904.71ms | 51083.22ms | **7.99** | 8.00 | **8.47** | 8.00 | **8.65** | 8.00 |
| Insertion Sort (O(n2)) | 0.97ms | 3.79ms | 15.77ms | 68.48ms | **3.92** | 8.00 | **4.16** | 8.00 | **4.34** | 8.00 |
| Descending Collection | Selection Sort (O(n2)) | 109.46ms | 876.80ms | 7415.57ms | 64740.74ms | **8.01** | 8.00 | **8.46** | 8.00 | **8.73** | 8.00 |
| Insertion Sort (O(n2)) | 85.00ms | 694.15ms | 5745.89ms | 50451.58ms | **8.17** | 8.00 | **8.28** | 8.00 | **8.78** | 8.00 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| **JavaLinkedList** | | N1 = 500 elements | N3 = 1000 elements | N4 = 2000 elements | N5 = 4000 elements | **Timerelation for N2 / N1** | O(n^2) + O(n^2) O-notation for N2 / N1 | **Timerelation for N3 / N2** | O(n^2) + O(n^2) O-notation for N3 / N2 | **Timerelation for N4 / N3** | O(n^2) + O(n^2) O-notation for N4 / N3 |  |
| T(N1) run 20 times | T(N2) run 10 times | T(N3) run 5 times | T(N4) run 2 times |  |
| Random Collection | Selection Sort (O(n2)) | 45.59ms | 382.00ms | 3643.14ms | 28707.58ms | **8.38** | 8.00 | **9.54** | 8.00 | **7.88** | 8.00 | Expected difference because sorting linked lists is O(n^2) for many of the operations and another O(n^2) for the sorting! hence TWICE the time to sort. variance in insertion sort efficiency for nearly sorted collections is O(n^2) + O(n) |
| Insertion Sort (O(n2)) | 36.92ms | 307.74ms | 2600.53ms | 23043.41ms | **8.34** | 8.00 | **8.45** | 8.00 | **8.86** | 8.00 |
| Nearly Sorted Collection | Selection Sort (O(n2)) | 40.18ms | 343.34ms | 3122.33ms | 25737.70ms | **8.55** | 8.00 | **9.09** | 8.00 | **8.24** | 8.00 |
| Insertion Sort (O(n2)) | 0.45ms | 1.78ms | 7.60ms | 36.89ms | **4.00** | 8.00 | **4.26** | 8.00 | **4.85** | 8.00 |
| Descending Collection | Selection Sort (O(n2)) | 47.28ms | 401.03ms | 3482.32ms | 30705.83ms | **8.48** | 8.00 | **8.68** | 8.00 | **8.82** | 8.00 |
| Insertion Sort (O(n2)) | 59.83ms | 502.28ms | 4497.81ms | 39771.24ms | **8.40** | 8.00 | **8.95** | 8.00 | **8.84** | 8.00 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| **JavaArrayList** | | N1 = 500 elements | N3 = 1000 elements | N4 = 2000 elements | N5 = 4000 elements | **Timerelation for N2 / N1** | O(n^2) O-notation for N2 / N1 | **Timerelation for N3 / N2** | O(n^2) O-notation for N3 / N2 | **Timerelation for N4 / N3** | O(n^2) O-notation for N4 / N3 |  |
| T(N1) run 20 times | T(N2) run 10 times | T(N3) run 5 times | T(N4) run 2 times |  |
| Random Collection | Selection Sort (O(n2)) | 2.28ms | 8.28ms | 37.59ms | 138.79ms | **3.64** | 4.00 | **4.54** | 4.00 | **3.69** | 4.00 |  |
| Insertion Sort (O(n2)) | 1.38ms | 5.30ms | 20.37ms | 81.50ms | **3.85** | 4.00 | **3.84** | 4.00 | **4.00** | 4.00 |  |
| Nearly Sorted Collection | Selection Sort (O(n2)) | 2.39ms | 8.83ms | 32.82ms | 123.35ms | **3.70** | 4.00 | **3.71** | 4.00 | **3.76** | 4.00 |  |
| Insertion Sort (O(n2)) | 0.03ms | 0.06ms | 0.11ms | 0.19ms | **1.80** | 4.00 | **1.84** | 4.00 | **1.69** | 4.00 | Variance happens because insertionSort on sorted arrays = O(n) instead of O(n^2)! |
| Descending Collection | Selection Sort (O(n2)) | 2.12ms | 8.86ms | 32.67ms | 132.52ms | **4.19** | 4.00 | **3.69** | 4.00 | **4.06** | 4.00 |
| Insertion Sort (O(n2)) | 2.61ms | 10.26ms | 41.85ms | 164.82ms | **3.93** | 4.00 | **4.08** | 4.00 | **3.94** | 4.00 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| **COLLECTION TESTS:** |  |  |  |  |  |  |  |  |  |  |  |  |
| **MyLinkedList** | | N1 = 5000 elements | N3 = 10000 elements | N4 = 20000 elements | N5 = 40000 elements | **Timerelation for N2 / N1** | O(n^2) || O(n) O-notation for N2 / N1 | **Timerelation for N3 / N2** | O(n^2) || O(n) O-notation for N3 / N2 | **Timerelation for N4 / N3** | O(n^2) || O(n) O-notation for N4 / N3 |  |
| T(N1) run 50 times | T(N2) run 50 times | T(N3) run 50 times | T(N4) run 50 times |  |
| Add (O(n^2)) | | 21.39ms | 98.43ms | 535.46ms | 2959.51ms | **4.60** | 4.00 | **5.44** | 4.00 | **5.53** | 4.00 |  |
| Get (O(n^2)) | | 20.78ms | 82.83ms | 338.52ms | 1359.67ms | **3.99** | 4.00 | **4.09** | 4.00 | **4.02** | 4.00 |  |
| AddFirst (O(n)) | | 0.22ms | 0.31ms | 0.63ms | 1.17ms | **1.41** | 2.00 | **2.04** | 2.00 | **1.87** | 2.00 |  |
| AddLast (O(n)) | | 0.22ms | 0.28ms | 0.67ms | 1.08ms | **1.29** | 2.00 | **2.39** | 2.00 | **1.63** | 2.00 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| **JavaLinkedList** | | N1 = 5000 elements | N3 = 10000 elements | N4 = 20000 elements | N5 = 40000 elements | **Timerelation for N2 / N1** | O(n^2) || O(n) O-notation for N2 / N1 | **Timerelation for N3 / N2** | O(n^2) || O(n) O-notation for N3 / N2 | **Timerelation for N4 / N3** | O(n^2) || O(n) O-notation for N4 / N3 |  |
| T(N1) run 50 times | T(N2) run 50 times | T(N3) run 50 times | T(N4) run 50 times |  |
| Add (O(n^2)) | | 9.88ms | 52.34ms | 313.07ms | 1682.11ms | **5.30** | 4.00 | **5.98** | 4.00 | **5.37** | 4.00 |  |
| Get (O(n^2)) | | 10.37ms | 43.66ms | 195.72ms | 867.87ms | **4.21** | 4.00 | **4.48** | 4.00 | **4.43** | 4.00 |
| AddFirst (O(n)) | | 0.20ms | 0.36ms | 1.05ms | 1.63ms | **1.84** | 2.00 | **2.88** | 2.00 | **1.55** | 2.00 |
| AddLast (O(n)) | | 0.21ms | 0.37ms | 0.85ms | 1.79ms | **1.79** | 2.00 | **2.31** | 2.00 | **2.11** | 2.00 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| **JavaArrayList** | | N1 = 5000 elements | N3 = 10000 elements | N4 = 20000 elements | N5 = 40000 elements | **Timerelation for N2 / N1** | O(n^2) || O(n) O-notation for N2 / N1 | **Timerelation for N3 / N2** | O(n^2) || O(n) O-notation for N3 / N2 | **Timerelation for N4 / N3** | O(n^2) || O(n) O-notation for N4 / N3 |  |
| T(N1) run 50 times | T(N2) run 50 times | T(N3) run 50 times | T(N4) run 50 times |  |
| Add (O(n^2)) | | 1.46ms | 5.19ms | 20.93ms | 84.21ms | **3.56** | 4.00 | **4.04** | 4.00 | **4.02** | 4.00 |  |
| Get (O(n)) | | 0.14ms | 0.23ms | 0.45ms | 1.00ms | **1.58** | 2.00 | **1.97** | 2.00 | **2.24** | 2.00 |  |
| AddFirst (O(n^2)) | | 2.79ms | 10.54ms | 43.24ms | 172.47ms | **3.77** | 4.00 | **4.10** | 4.00 | **3.99** | 4.00 |  |
| AddLast (O(n)) | | 0.15ms | 0.26ms | 0.54ms | 1.18ms | **1.73** | 2.00 | **2.08** | 2.00 | **2.16** | 2.00 |  |

## Java:

package Execute;

import efficiency.CollectionEfficiencyTest;

import efficiency.SortingEfficiencyTest;

public class RunTests

{

public static void main(String[] args)

{

System.err.println("SORTING");

new SortingEfficiencyTest().runTest();

System.err.println("");

System.err.println("COLLECTION");

new CollectionEfficiencyTest().runTest();

System.err.println("");

}

}

package Helpers;

public class arrayProperties

{

public int numberOfTimesToRun = 1;

public int sizeOfSortArray = 20;

public int testFromZeroTo = 20;

}

package Sorting;

import collection.MyCollection;

import collection.MyLinkedList;

public class BuiltInQuickSort implements Sortable

{

@Override

public MyCollection sort(MyCollection arrayToSort)

{

int[] sorted = arrayToSort.toArray();

java.util.Arrays.sort(sorted);

return new MyLinkedList().createFromArray(sorted);

}

}

package Sorting;

import collection.MyCollection;

public class InsertionSort implements Sortable

{

private MyCollection arrayToSearch;

private int currentSortIndex;

private int currentInsertElementID;

@Override

public MyCollection sort(MyCollection arrayToSort)

{

arrayToSearch = arrayToSort;

currentSortIndex = 1;

for (; currentSortIndex < arrayToSearch.size(); ++currentSortIndex)

{

shiftElements();

}

return arrayToSearch;

}

private void shiftElements()

{

currentInsertElementID = currentSortIndex;

boolean keepShifting = isCurrentElementSmallerThanPrevious();

while (keepShifting && currentInsertElementID > 0)

{

keepShifting = isCurrentElementSmallerThanPrevious();

if (keepShifting)

{

shiftOneElementDown();

--currentInsertElementID;

}

}

}

private boolean isCurrentElementSmallerThanPrevious()

{

int previousElement = arrayToSearch.get(currentInsertElementID - 1).getContents();

int currentElement = arrayToSearch.get(currentInsertElementID).getContents();

return currentElement < previousElement;

}

private void shiftOneElementDown()

{

int smallElementPlaceholder = arrayToSearch.get(currentInsertElementID).getContents();

int largeELementPlaceholder = arrayToSearch.get(currentInsertElementID - 1).getContents();

arrayToSearch.update(largeELementPlaceholder, currentInsertElementID);

arrayToSearch.update(smallElementPlaceholder, currentInsertElementID - 1);

}

}

/\*

\* 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 Sorting;

import collection.MyCollection;

/\*\*

\*

\* @author James

\*/

public class InsertionSortFSM implements Sortable

{

private MyCollection arrayToSearch;

private int currentSortIndex;

private int currentExtractedElement;

private int openSpaceIndex;

@Override

public MyCollection sort(MyCollection arrayToSort)

{

arrayToSearch = arrayToSort;

currentSortIndex = 1;

for (; currentSortIndex < arrayToSearch.size(); ++currentSortIndex)

{

extractElement();

if (!testElement())

{

shiftTestContinous();

}

insertElementToOpen();

}

return arrayToSearch;

}

private void extractElement()

{

currentExtractedElement = arrayToSearch.get(currentSortIndex).getContents();

openSpaceIndex = currentSortIndex;

}

private boolean testElement()

{

if (openSpaceIndex > 0)

{

return currentExtractedElement >= arrayToSearch.get(openSpaceIndex - 1).getContents();

}

return true;

}

private void shiftTestContinous()

{

do

{

shiftElement();

} while (!testElement() && openSpaceIndex > 0);

}

private void shiftElement()

{

int elementToShift = arrayToSearch.get(openSpaceIndex - 1).getContents();

arrayToSearch.update(elementToShift, openSpaceIndex);

--openSpaceIndex;

}

private void insertElementToOpen()

{

arrayToSearch.update(currentExtractedElement, openSpaceIndex);

}

}

package Sorting;

import collection.MyCollection;

public class SelectionSort implements Sortable

{

private MyCollection listToSearch;

private int currentSortedIndex;

private int currentSmallestElementID;

@Override

public MyCollection sort(MyCollection arrayToSort)

{

currentSortedIndex = 0;

listToSearch = arrayToSort;

for (; currentSortedIndex < listToSearch.size() - 1; ++currentSortedIndex)

{

searchSmallestElementID();

swapElements();

}

return listToSearch;

}

private void searchSmallestElementID()

{

currentSmallestElementID = currentSortedIndex;

for (int i = currentSortedIndex; i < listToSearch.size(); ++i)

{

testIfSmallerAndStoreIndex(listToSearch.get(currentSmallestElementID).getContents(), listToSearch.get(i).getContents(), i);

}

}

private void testIfSmallerAndStoreIndex(int storedElement, int testElement, int testIndex)

{

if (testElement <= storedElement)

{

currentSmallestElementID = testIndex;

}

}

private void swapElements()

{

int elementToInsert = listToSearch.get(currentSmallestElementID).getContents();

int elementAtCurrentRoot = listToSearch.get(currentSortedIndex).getContents();

listToSearch.update(elementAtCurrentRoot, currentSmallestElementID);

listToSearch.update(elementToInsert, currentSortedIndex);

}

}

/\*

\* 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 Sorting;

import collection.MyCollection;

/\*\*

\*

\* @author JamesFoxes

\*/

public interface Sortable

{

public MyCollection sort(MyCollection arrayToSort);

}

/\*

\* 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 collection;

/\*\*

\*

\* @author JamesFoxes

\*/

public enum CollectionType

{

MyLinkedList, JavaLinkedList, Array

}

/\*

\* 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 collection;

import collection.nodes.BetweenNode;

import java.util.ArrayList;

/\*\*

\*

\* @author JamesFoxes

\*/

public class MyArrayList implements MyCollection

{

ArrayList<Integer> arrayList;

public MyArrayList()

{

arrayList = new ArrayList<>();

}

@Override

public void addFirst(int dataToStore)

{

arrayList.add(0, dataToStore);

}

@Override

public void addLast(int dataToStore)

{

arrayList.add(arrayList.size(), dataToStore);

}

@Override

public void add(int dataToStore, int index)

{

arrayList.add(index, dataToStore);

}

@Override

public BetweenNode get(int index)

{

return new BetweenNode(arrayList.get(index));

}

@Override

public int size()

{

return arrayList.size();

}

@Override

public void update(int dataToAdd, int index)

{

arrayList.set(index, dataToAdd);

}

@Override

public int[] toArray()

{

int[] toReturn = new int[arrayList.size()];

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

{

toReturn[i] = arrayList.get(i);

}

return toReturn;

}

@Override

public MyCollection createFromArray(int[] array)

{

MyCollection toReturn = new MyArrayList();

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

{

toReturn.add(array[i], i);

}

return toReturn;

}

@Override

public void printArrayInLine()

{

for (Integer arrayListElement : arrayList)

{

System.out.print(arrayListElement);

}

}

@Override

public BetweenNode getFirst()

{

return new BetweenNode(arrayList.get(0));

}

@Override

public BetweenNode getLast()

{

return new BetweenNode(arrayList.get(arrayList.size() - 1));

}

@Override

public void removeFirst()

{

arrayList.remove(0);

}

@Override

public void removeLast()

{

arrayList.remove(arrayList.size() - 1);

}

@Override

public void remove(int index)

{

arrayList.remove(index);

}

}

/\*

\* 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 collection;

import collection.nodes.BetweenNode;

/\*\*

\*

\* @author JamesFoxes

\*/

public interface MyCollection

{

public void addFirst(int dataToStore);

public void addLast(int dataToStore);

public void add(int dataToStore, int index);

public BetweenNode get(int index);

public int size();

public void update(int dataToAdd, int index);

public int[] toArray();

public MyCollection createFromArray(int[] array);

public void printArrayInLine();

public BetweenNode getFirst();

public BetweenNode getLast();

public void removeFirst();

public void removeLast();

public void remove(int index);

}

/\*

\* 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 collection;

import collection.nodes.BetweenNode;

import java.util.LinkedList;

/\*\*

\*

\* @author JamesFoxes

\*/

public class MyJavaLinkedList implements MyCollection

{

LinkedList<Integer> linkedList;

public MyJavaLinkedList()

{

linkedList = new LinkedList<>();

}

@Override

public void addFirst(int dataToStore)

{

linkedList.addFirst(dataToStore);

}

@Override

public void addLast(int dataToStore)

{

linkedList.addLast(dataToStore);

}

@Override

public void add(int dataToStore, int index)

{

linkedList.add(index, dataToStore);

}

@Override

public BetweenNode get(int index)

{

return new BetweenNode(linkedList.get(index));

}

@Override

public int size()

{

return linkedList.size();

}

@Override

public void update(int dataToAdd, int index)

{

linkedList.set(index, dataToAdd);

}

@Override

public int[] toArray()

{

int[] toReturn = new int[linkedList.size()];

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

{

toReturn[i] = linkedList.get(i);

}

return toReturn;

}

@Override

public MyCollection createFromArray(int[] array)

{

MyCollection toReturn = new MyArrayList();

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

{

toReturn.add(array[i], i);

}

return toReturn;

}

@Override

public void printArrayInLine()

{

for (Integer arrayListElement : linkedList)

{

System.out.print(arrayListElement);

}

}

@Override

public BetweenNode getFirst()

{

return new BetweenNode(linkedList.getFirst());

}

@Override

public BetweenNode getLast()

{

return new BetweenNode(linkedList.getLast());

}

@Override

public void removeFirst()

{

linkedList.removeFirst();

}

@Override

public void removeLast()

{

linkedList.removeLast();

}

@Override

public void remove(int index)

{

linkedList.remove(index);

}

}

/\*

\* 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 collection;

import collection.nodes.BetweenNode;

import collection.nodes.SentinelNode;

import collection.nodes.Node;

/\*\*

\*

\* @author James

\*/

public class MyLinkedList implements MyCollection

{

private int size;

private SentinelNode firstNode;

private SentinelNode lastNode;

public MyLinkedList()

{

initializeList();

}

private void initializeList()

{

size = 0;

firstNode = new SentinelNode();

lastNode = new SentinelNode();

firstNode.setNextNode(lastNode);

lastNode.setPreviousNode(firstNode);

}

@Override

public int size()

{

return size;

}

public boolean isEmpty()

{

return size == 0;

}

@Override

public void addLast(int dataToStore)

{

lastNode.addNodeBefore(dataToStore);

++size;

}

@Override

public void addFirst(int dataToStore)

{

firstNode.addNodeAfter(dataToStore);

++size;

}

@Override

public void removeLast()

{

BetweenNode toRemove = tryToCastNode(lastNode.getPreviousNode());

toRemove.removeNode();

--size;

}

@Override

public void removeFirst()

{

BetweenNode toRemove = tryToCastNode(firstNode.getNextNode());

toRemove.removeNode();

--size;

}

@Override

public BetweenNode getFirst()

{

return tryToCastNode(firstNode.getNextNode());

}

@Override

public BetweenNode getLast()

{

return tryToCastNode(lastNode.getPreviousNode());

}

private BetweenNode tryToCastNode(Node toCast)

{

BetweenNode cast;

try

{

cast = (BetweenNode) toCast;

} catch (ClassCastException e)

{

System.err.println("Array was empty");

initializeList();

return null;

}

return cast;

}

@Override

public void add(int dataToStore, int index)

{

Node iterator = firstNode;

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

{

iterator = iterator.getNextNodeForced();

}

iterator.addNodeAfter(dataToStore);

++size;

}

@Override

public BetweenNode get(int index)

{

Node iterator = firstNode.getNextNode();

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

{

if (iterator instanceof SentinelNode)

{

throw new NullPointerException("No such element");

}

iterator = iterator.getNextNode();

}

return (BetweenNode) iterator;

}

@Override

public void update(int dataToAdd, int index)

{

BetweenNode toUpdate = get(index);

toUpdate.setContents(dataToAdd);

}

@Override

public void remove(int index)

{

BetweenNode toRemove = get(index);

toRemove.removeNode();

--size;

}

public int findFirst(int contentToFind)

{

Node iterator = firstNode;

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

{

iterator = iterator.getNextNode();

if(iterator instanceof BetweenNode && iterator.getContents() == contentToFind)

{

return i;

}

}

return -1;

}

public int[] findAll(int contentToFind)

{

Node iterator = firstNode;

int[] indicies = new int[size];

int amountFound = 0;

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

{

iterator = iterator.getNextNode();

if(iterator instanceof BetweenNode && iterator.getContents() == contentToFind)

{

indicies[amountFound++] = iterator.getContents();

}

}

return indicies;

}

public void clear()

{

initializeList();

}

@Override

public int[] toArray()

{

int[] toReturn = new int[size];

Node iterator = firstNode;

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

{

iterator = iterator.getNextNode();

toReturn[i] = iterator.getContents();

}

return toReturn;

}

@Override

public MyCollection createFromArray(int[] array)

{

initializeList();

for (int i = 0; i < array.length; ++i)

{

add(array[i], i);

}

return this;

}

@Override

public void printArrayInLine()

{

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

{

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

}

}

}

/\*

\* 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 collection.nodes;

/\*\*

\*

\* @author James

\*/

public class BetweenNode extends Node

{

private int contents;

public BetweenNode(int contents)

{

this.contents = contents;

}

@Override

public int getContents()

{

return contents;

}

@Override

public void setContents(int dataToSet)

{

contents = dataToSet;

}

public void removeNode()

{

nextNode.setPreviousNode(previousNode);

previousNode.setNextNode(nextNode);

}

}

/\*

\* 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 collection.nodes;

/\*\*

\*

\* @author JamesFoxes

\*/

public abstract class Node

{

protected Node previousNode;

protected Node nextNode;

public void setNextNode(Node node)

{

nextNode = node;

}

public void setPreviousNode(Node node)

{

previousNode = node;

}

public void addNodeBefore(int dataToAdd)

{

Node toAdd = new BetweenNode(dataToAdd);

toAdd.setPreviousNode(previousNode);

previousNode.setNextNode(toAdd);

previousNode = toAdd;

toAdd.setNextNode(this);

}

public void addNodeAfter(int dataToAdd)

{

Node toAdd = new BetweenNode(dataToAdd);

toAdd.setNextNode(nextNode);

nextNode.setPreviousNode(toAdd);

nextNode = toAdd;

toAdd.setPreviousNode(this);

}

public Node getNextNode()

{

return nextNode;

}

public Node getNextNodeForced()

{

if(nextNode == null)

{

addNodeAfter(0);

}

return nextNode;

}

public Node getPreviousNode()

{

return previousNode;

}

public abstract int getContents();

public abstract void setContents(int dataToSet);

}

/\*

\* 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 collection.nodes;

/\*\*

\*

\* @author JamesFoxes

\*/

public class SentinelNode extends Node

{

@Override

public int getContents()

{

return 0;

}

@Override

public void setContents(int dataToSet)

{

}

}

/\*

\* 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 efficiency;

import collection.MyJavaLinkedList;

import collection.\*;

/\*\*

\*

\* @author JamesFoxes

\*/

public class CollectionEfficiencyTest

{

CollectionTestResults testResults;

int numberOfElements;

int timesToRun = 200;

private long startTime = 0;

public void runTest()

{

System.out.println("--- My LinkedList efficiency test ---");

System.out.println("-5000 elements-");

runAll(timesToRun, 5000, CollectionType.MyLinkedList);

printResults();

System.out.println("-10000 elements-");

runAll(timesToRun, 10000, CollectionType.MyLinkedList);

printResults();

System.out.println("-20000 elements-");

runAll(timesToRun, 20000, CollectionType.MyLinkedList);

printResults();

System.out.println("-40000 elements-");

runAll(timesToRun, 40000, CollectionType.MyLinkedList);

printResults();

System.out.println("--- Java LinkedList efficiency test ---");

System.out.println("-5000 elements-");

runAll(timesToRun, 5000, CollectionType.JavaLinkedList);

printResults();

System.out.println("-10000 elements-");

runAll(timesToRun, 10000, CollectionType.JavaLinkedList);

printResults();

System.out.println("-20000 elements-");

runAll(timesToRun, 20000, CollectionType.JavaLinkedList);

printResults();

System.out.println("-40000 elements-");

runAll(timesToRun, 40000, CollectionType.JavaLinkedList);

printResults();

System.out.println("--- Java ArrayList efficiency test ---");

System.out.println("-5000 elements-");

runAll(timesToRun, 5000, CollectionType.Array);

printResults();

System.out.println("-10000 elements-");

runAll(timesToRun, 10000, CollectionType.Array);

printResults();

System.out.println("-20000 elements-");

runAll(timesToRun, 20000, CollectionType.Array);

printResults();

System.out.println("-40000 elements-");

runAll(timesToRun, 40000, CollectionType.Array);

printResults();

}

private void runAll(int numberOfTimes, int numberOfElements, CollectionType type)

{

this.numberOfElements = numberOfElements;

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

{

testResults = new CollectionTestResults(numberOfElements);

runAllOnce(type);

testResults.incrementTimesRun();

}

}

private void runAllOnce(CollectionType type)

{

testAdd(type);

testGet(type);

testAddFirst(type);

testAddLast(type);

}

public void testAdd(CollectionType type)

{

MyCollection toTest = createCollection(type);

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

{

startTimer();

toTest.add(i, i);

stopTimerAndSaveElapsedTime(Operation.add);

}

}

public void testGet(CollectionType type)

{

MyCollection toTest = createCollection(type);

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

{

toTest.add(i, i);

}

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

{

startTimer();

toTest.get(i);

stopTimerAndSaveElapsedTime(Operation.get);

}

}

public void testAddFirst(CollectionType type)

{

MyCollection toTest = createCollection(type);

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

{

startTimer();

toTest.addFirst(i);

stopTimerAndSaveElapsedTime(Operation.addFirst);

}

}

public void testAddLast(CollectionType type)

{

MyCollection toTest = createCollection(type);

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

{

startTimer();

toTest.addLast(i);

stopTimerAndSaveElapsedTime(Operation.addLast);

}

}

private MyCollection createCollection(CollectionType type)

{

MyCollection testCollection;

switch (type)

{

case MyLinkedList:

testCollection = new MyLinkedList();

break;

case JavaLinkedList:

testCollection = new MyJavaLinkedList();

break;

case Array:

testCollection = new MyArrayList();

break;

default:

testCollection = null;

System.err.println("ERROR: createCollection!");

break;

}

return testCollection;

}

private void startTimer()

{

startTime = System.nanoTime();

}

private void stopTimerAndSaveElapsedTime(Operation operation)

{

switch (operation)

{

case add:

testResults.addTime += System.nanoTime() - startTime;

break;

case get:

testResults.getTime += System.nanoTime() - startTime;

break;

case addFirst:

testResults.addFirstTime += System.nanoTime() - startTime;

break;

case addLast:

testResults.addLastTime += System.nanoTime() - startTime;

break;

}

}

private void printResults()

{

testResults.printAverageTimes();

}

}

enum Operation

{

add, get, addFirst, addLast

}

/\*

\* 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 efficiency;

/\*\*

\*

\* @author JamesFoxes

\*/

class CollectionTestResults

{

public long addTime;

public long getTime;

public long addFirstTime;

public long addLastTime;

private int elementsRun;

private int timesRun = 0;

public CollectionTestResults(int elementsRun)

{

this.elementsRun = elementsRun;

}

public void incrementTimesRun()

{

++timesRun;

}

public void printAverageTimes()

{

System.out.println("Time: " + ((double) addTime) / (1000000 \* timesRun) + "ms to add " + elementsRun + " elements. With an average of: " + getAverageMilisecondTime(addTime) + "ms per element.");

System.out.println("Time: " + ((double) getTime) / (1000000 \* timesRun) + "ms to get " + elementsRun + " elements. With an average of: " + getAverageMilisecondTime(getTime) + "ms per element.");

System.out.println("Time: " + ((double) addFirstTime) / (1000000 \* timesRun) + "ms to add " + elementsRun + " elements first. With an average of: " + getAverageMilisecondTime(addFirstTime) + "ms per element.");

System.out.println("Time: " + ((double) addLastTime) / (1000000 \* timesRun) + "ms to add " + elementsRun + " elements last. With an average of: " + getAverageMilisecondTime(addLastTime) + "ms per element.");

}

private double getAverageMilisecondTime(long fullDuration)

{

long averageTime = fullDuration / (elementsRun \* timesRun);

double averageTimeMiliseconds = ((double) averageTime) / 1000000;

return averageTimeMiliseconds;

}

}

/\*

\* 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 efficiency;

import Helpers.arrayProperties;

import Sorting.BuiltInQuickSort;

import Sorting.InsertionSortFSM;

import Sorting.SelectionSort;

import Sorting.Sortable;

import collection.CollectionType;

/\*\*

\*

\* @author JamesFoxes

\*/

public class SortingEfficiencyTest

{

public void runTest()

{

System.out.println("Selection sorting --->");

runOneSortMethod(new SelectionSort());

System.out.println("Insertion sorting --->");

runOneSortMethod(new InsertionSortFSM());

System.out.println("Quick sorting --->");

runOneSortMethod(new BuiltInQuickSort());

}

private void runOneSortMethod(Sortable sortingMethod)

{

System.out.println("My LinkedList");

executeTestsAndPrintAverageTime(createTestEnvironment(1000), sortingMethod, CollectionType.MyLinkedList);

executeTestsAndPrintAverageTime(createTestEnvironment(2000), sortingMethod, CollectionType.MyLinkedList);

executeTestsAndPrintAverageTime(createTestEnvironment(4000), sortingMethod, CollectionType.MyLinkedList);

System.out.println("Java LinkedList");

executeTestsAndPrintAverageTime(createTestEnvironment(1000), sortingMethod, CollectionType.JavaLinkedList);

executeTestsAndPrintAverageTime(createTestEnvironment(2000), sortingMethod, CollectionType.JavaLinkedList);

executeTestsAndPrintAverageTime(createTestEnvironment(4000), sortingMethod, CollectionType.JavaLinkedList);

System.out.println("Java ArrayList");

executeTestsAndPrintAverageTime(createTestEnvironment(1000), sortingMethod, CollectionType.Array);

executeTestsAndPrintAverageTime(createTestEnvironment(2000), sortingMethod, CollectionType.Array);

executeTestsAndPrintAverageTime(createTestEnvironment(4000), sortingMethod, CollectionType.Array);

}

private arrayProperties createTestEnvironment(int arraySize)

{

arrayProperties test = new arrayProperties();

test.sizeOfSortArray = arraySize;

test.numberOfTimesToRun = (100000 / arraySize);

test.testFromZeroTo = arraySize \* 5;

return test;

}

private void executeTestsAndPrintAverageTime(arrayProperties arrayProperties, Sortable sortingMethod, CollectionType collectionType)

{

long[] executionTimes = new SortingTester(arrayProperties, collectionType).run(sortingMethod);

long averageTime = executionTimes[0] / arrayProperties.numberOfTimesToRun;

double averageTimeMiliseconds = ((double) averageTime) / 1000000;

System.out.println("Scrambled arrays: Average Runtime of " + averageTimeMiliseconds + "ms, average over " + arrayProperties.numberOfTimesToRun + " runs with " + arrayProperties.sizeOfSortArray + " elements in each array.");

averageTime = executionTimes[1] / arrayProperties.numberOfTimesToRun;

averageTimeMiliseconds = ((double) averageTime) / 1000000;

System.out.println("Nearly Sorted arrays: Average Runtime of " + averageTimeMiliseconds + "ms, average over " + arrayProperties.numberOfTimesToRun + " runs with " + arrayProperties.sizeOfSortArray + " elements in each array.");

averageTime = executionTimes[2] / arrayProperties.numberOfTimesToRun;

averageTimeMiliseconds = ((double) averageTime) / 1000000;

System.out.println("Descending arrays: Average Runtime of " + averageTimeMiliseconds + "ms, average over " + arrayProperties.numberOfTimesToRun + " runs with " + arrayProperties.sizeOfSortArray + " elements in each array.");

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

}

}

package efficiency;

import Helpers.\*;

import Sorting.\*;

import collection.CollectionType;

import collection.MyArrayList;

import collection.MyCollection;

import collection.MyLinkedList;

import java.util.ArrayList;

public class SortingTester

{

private final int numberOfTimesToRun;

private final int arraySize;

private final int testFromZeroTo;

private MyCollection scrambledArray;

private MyCollection nearlySortedArray;

private MyCollection descendingArray;

private volatile long startTime;

private volatile long[] elapsedTimes;

private Sortable sortMethod;

private CollectionType arrayType;

public SortingTester(arrayProperties arrayProperties, CollectionType arrayType)

{

elapsedTimes = new long[3];

this.arrayType = arrayType;

numberOfTimesToRun = arrayProperties.numberOfTimesToRun;

arraySize = arrayProperties.sizeOfSortArray;

testFromZeroTo = arrayProperties.testFromZeroTo;

}

public long[] run(Sortable sortMethod)

{

this.sortMethod = sortMethod;

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

{

createRandomArray();

createNearlySortedArray();

createDescendingArray();

doSort(Method.scrambledArray, scrambledArray);

doSort(Method.nearlySortedArray, nearlySortedArray);

doSort(Method.descendingArray, descendingArray);

}

return elapsedTimes;

}

private void createRandomArray()

{

switch (arrayType)

{

case MyLinkedList:

scrambledArray = new MyLinkedList();

break;

case Array:

scrambledArray = new MyArrayList();

break;

}

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

{

scrambledArray.add((int) (Math.random() \* testFromZeroTo), i);

}

}

private void createNearlySortedArray()

{

switch (arrayType)

{

case MyLinkedList:

nearlySortedArray = new MyLinkedList();

break;

case Array:

nearlySortedArray = new MyArrayList();

break;

}

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

{

nearlySortedArray.add(i, i);

}

int chunkSize = 4;

for (int i = 0; i < arraySize; i += chunkSize)

{

int cap = Math.min(chunkSize + i, arraySize);

for (int j = i; j < (cap - 1); ++j)

{

int swapTarget = (int) (Math.random() \* (cap - j)) + j;

swapElements(nearlySortedArray, j, swapTarget);

}

}

}

private void swapElements(MyCollection array, int from, int to)

{

int fromContents = array.get(from).getContents();

int toContents = array.get(to).getContents();

array.update(fromContents, to);

array.update(toContents, from);

}

private void createDescendingArray()

{

switch (arrayType)

{

case MyLinkedList:

descendingArray = new MyLinkedList();

break;

case Array:

descendingArray = new MyArrayList();

break;

}

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

{

descendingArray.add(arraySize - i, i);

}

}

private void doSort(Method method, MyCollection arrayToSort)

{

startTimer();

sortMethod.sort(arrayToSort);

stopTimerAndSaveElapsedTime(method);

}

private void startTimer()

{

startTime = System.nanoTime();

}

private void stopTimerAndSaveElapsedTime(Method method)

{

switch (method)

{

case scrambledArray:

elapsedTimes[0] += System.nanoTime() - startTime;

break;

case nearlySortedArray:

elapsedTimes[1] += System.nanoTime() - startTime;

break;

case descendingArray:

elapsedTimes[2] += System.nanoTime() - startTime;

break;

}

}

public ArrayList<MyCollection> returnSortedArraysForTesting(Sortable sortMethod)

{

this.sortMethod = sortMethod;

ArrayList<MyCollection> returnArrays = new ArrayList<>();

createRandomArray();

createNearlySortedArray();

createDescendingArray();

doSort(Method.scrambledArray, scrambledArray);

doSort(Method.nearlySortedArray, nearlySortedArray);

doSort(Method.descendingArray, descendingArray);

returnArrays.add(scrambledArray);

returnArrays.add(nearlySortedArray);

returnArrays.add(descendingArray);

return returnArrays;

}

public ArrayList<MyCollection> returnComparisonArraysForTesting(ArrayList<MyCollection> arrays)

{

ArrayList<MyCollection> returnArrays = new ArrayList<>();

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

{

int[] sortedArrayEntry = arrays.get(i).toArray();

java.util.Arrays.sort(sortedArrayEntry);

returnArrays.add(new MyLinkedList().createFromArray(sortedArrayEntry));

}

return returnArrays;

}

private void printArray(int[] array)

{

for (int element : array)

{

System.out.println(Integer.toString(element));

}

}

}

enum Method

{

scrambledArray, nearlySortedArray, descendingArray

}

package Execute;

import efficiency.SortingTester;

import Helpers.arrayProperties;

import Sorting.InsertionSort;

import Sorting.InsertionSortFSM;

import Sorting.SelectionSort;

import collection.CollectionType;

import collection.MyCollection;

import java.util.ArrayList;

import org.junit.Test;

import static org.junit.Assert.\*;

public class TesterTest

{

@Test

public void SelectionSortTest()

{

arrayProperties shortTest = new arrayProperties();

SortingTester instance = new SortingTester(shortTest, CollectionType.MyLinkedList);

shortTest.sizeOfSortArray = 1000;

shortTest.numberOfTimesToRun = 1;

shortTest.testFromZeroTo = 5000;

ArrayList<MyCollection> result = instance.returnSortedArraysForTesting(new SelectionSort());

ArrayList<MyCollection> expResult = instance.returnComparisonArraysForTesting(result);

assertArrayEquals(expResult.get(0).toArray(), result.get(0).toArray());

assertArrayEquals(expResult.get(1).toArray(), result.get(1).toArray());

assertArrayEquals(expResult.get(2).toArray(), result.get(2).toArray());

}

@Test

public void InsertionSortTest()

{

arrayProperties shortTest = new arrayProperties();

SortingTester instance = new SortingTester(shortTest, CollectionType.MyLinkedList);

shortTest.sizeOfSortArray = 1000;

shortTest.numberOfTimesToRun = 1;

shortTest.testFromZeroTo = 5000;

ArrayList<MyCollection> result = instance.returnSortedArraysForTesting(new InsertionSort());

ArrayList<MyCollection> expResult = instance.returnComparisonArraysForTesting(result);

assertArrayEquals(expResult.get(0).toArray(), result.get(0).toArray());

assertArrayEquals(expResult.get(1).toArray(), result.get(1).toArray());

assertArrayEquals(expResult.get(2).toArray(), result.get(2).toArray());

}

@Test

public void InsertionSortFSMTest()

{

arrayProperties shortTest = new arrayProperties();

SortingTester instance = new SortingTester(shortTest, CollectionType.MyLinkedList);

shortTest.sizeOfSortArray = 1000;

shortTest.numberOfTimesToRun = 1;

shortTest.testFromZeroTo = 5000;

ArrayList<MyCollection> result = instance.returnSortedArraysForTesting(new InsertionSortFSM());

ArrayList<MyCollection> expResult = instance.returnComparisonArraysForTesting(result);

assertArrayEquals(expResult.get(0).toArray(), result.get(0).toArray());

assertArrayEquals(expResult.get(1).toArray(), result.get(1).toArray());

assertArrayEquals(expResult.get(2).toArray(), result.get(2).toArray());

}

}

package Sorting;

import efficiency.SortingTester;

import Helpers.arrayProperties;

import collection.CollectionType;

import collection.MyCollection;

import collection.MyLinkedList;

import java.util.ArrayList;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class SortingTest

{

MyCollection arrayToSort;

private MyCollection createRandomArray(int arraySize, int fromZeroTo)

{

MyCollection scrambledArray = new MyLinkedList();

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

{

scrambledArray.add((int) (Math.random() \* fromZeroTo), i);

}

return scrambledArray;

}

@Before

public void setUp()

{

arrayToSort = createRandomArray(500, 1000);

System.out.println("Printing Array to be sorted: (size is " + arrayToSort.size() + ")");

arrayToSort.printArrayInLine();

System.out.println("");

}

@Test

public void testSelectionSort()

{

MyCollection result = new SelectionSort().sort(arrayToSort);

ArrayList<MyCollection> toBeQuickSorted = new ArrayList<>();

toBeQuickSorted.add(arrayToSort);

MyCollection expected = new SortingTester(new arrayProperties(), CollectionType.MyLinkedList).returnComparisonArraysForTesting(toBeQuickSorted).get(0);

assertArrayEquals(expected.toArray(), result.toArray());

}

@Test

public void testInsertionSort()

{

MyCollection result = new InsertionSort().sort(arrayToSort);

ArrayList<MyCollection> toBeQuickSorted = new ArrayList<>();

toBeQuickSorted.add(arrayToSort);

MyCollection expected = new SortingTester(new arrayProperties(), CollectionType.MyLinkedList).returnComparisonArraysForTesting(toBeQuickSorted).get(0);

assertArrayEquals(expected.toArray(), result.toArray());

}

@Test

public void testInsertionFSMSort()

{

MyCollection result = new InsertionSortFSM().sort(arrayToSort);

ArrayList<MyCollection> toBeQuickSorted = new ArrayList<>();

toBeQuickSorted.add(arrayToSort);

MyCollection expected = new SortingTester(new arrayProperties(), CollectionType.MyLinkedList).returnComparisonArraysForTesting(toBeQuickSorted).get(0);

assertArrayEquals(expected.toArray(), result.toArray());

}

@Test

public void testBuiltInQuickSort()

{

MyCollection result = new BuiltInQuickSort().sort(arrayToSort);

ArrayList<MyCollection> toBeQuickSorted = new ArrayList<>();

toBeQuickSorted.add(arrayToSort);

MyCollection expected = new SortingTester(new arrayProperties(), CollectionType.MyLinkedList).returnComparisonArraysForTesting(toBeQuickSorted).get(0);

assertArrayEquals(expected.toArray(), result.toArray());

}

}

/\*

\* 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 mylinkedlist;

import collection.MyCollection;

import collection.MyLinkedList;

import mylinkedlist.MyLinkedListTest.OperationType;

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.Test;

/\*\*

\*

\* @author James

\*/

public class MyLinkedListTest

{

MyCollection testList;

@Before

public void setUp()

{

testList = new MyLinkedList();

initializeIncrementingList();

}

private void initializeIncrementingList()

{

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

{

testList.add(i, i);

}

}

private void mySizeAssertion(int expected)

{

assertEquals(expected, testList.size());

}

@Test

public void testAddFirst()

{

mySizeAssertion(10);

int dataToAdd = 5;

testList.addFirst(dataToAdd);

assertEquals(dataToAdd, testList.getFirst().getContents());

mySizeAssertion(11);

}

@Test

public void testAddLast()

{

mySizeAssertion(10);

int dataToAdd = 5;

testList.addLast(dataToAdd);

assertEquals(dataToAdd, testList.getLast().getContents());

mySizeAssertion(11);

}

@Test

public void testRemoveFirst()

{

assertEquals(0, testList.getFirst().getContents());

testList.removeFirst();

assertEquals(1, testList.getFirst().getContents());

}

@Test

public void testRemoveLast()

{

assertEquals(9, testList.getLast().getContents());

testList.removeLast();

assertEquals(8, testList.getLast().getContents());

}

@Test

public void testGetFirst()

{

assertEquals(0, testList.getFirst().getContents());

}

@Test

public void testGetLast()

{

assertEquals(9, testList.getLast().getContents());

}

@Test

public void testAdd()

{

mySizeAssertion(10);

int dataToAdd = 5;

testList.add(dataToAdd, 5);

mySizeAssertion(11);

assertEquals(5, testList.get(5).getContents());

assertEquals(5, testList.get(6).getContents());

assertEquals(9, testList.get(10).getContents());

}

@Test

public void testUpdate()

{

mySizeAssertion(10);

int dataToAdd = 5;

testList.update(dataToAdd, 1);

mySizeAssertion(10);

assertEquals(5, testList.get(1).getContents());

}

@Test

public void testRemove()

{

mySizeAssertion(10);

testList.remove(2);

mySizeAssertion(9);

assertEquals(1, testList.get(1).getContents());

assertEquals(3, testList.get(2).getContents());

assertEquals(4, testList.get(3).getContents());

}

@Test

public void testSizeSimpleAdd()

{

int amountOfElementsToAdd = 42;

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

{

testList.addLast(i);

}

assertEquals(amountOfElementsToAdd + 10, testList.size());

}

@Test

public void testSizeRandomOperations()

{

int amountOfOperations = 42;

int amountOfElements = 0;

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

{

switch (chooseRandomOperation())

{

case addFirst:

++amountOfElements;

testList.addFirst(i);

break;

case addLast:

++amountOfElements;

testList.addLast(i);

break;

}

}

amountOfOperations = 20;

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

{

switch (chooseRandomOperation())

{

case addFirst:

++amountOfElements;

testList.addFirst(i);

break;

case addLast:

++amountOfElements;

testList.addLast(i);

break;

case removeLast:

--amountOfElements;

testList.removeLast();

break;

case removeFirst:

--amountOfElements;

testList.removeFirst();

break;

}

}

assertEquals(amountOfElements + 10, testList.size());

}

private OperationType chooseRandomOperation()

{

OperationType[] types = OperationType.values();

int randomInt = (int) Math.floor(3 \* Math.random());

return types[randomInt];

}

enum OperationType

{

addFirst, addLast, removeLast, removeFirst

}

}

package suite;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

@RunWith(Suite.class)

@Suite.SuiteClasses(

{

Sorting.SortingTest.class,

Execute.TesterTest.class,

mylinkedlist.MyLinkedListTest.class

})

public class SortingTestsSuite

{

@BeforeClass

public static void setUpClass() throws Exception

{

}

@AfterClass

public static void tearDownClass() throws Exception

{

}

@Before

public void setUp() throws Exception

{

}

@After

public void tearDown() throws Exception

{

}

}