Program 5 : 50 points: LinkedLists/User Defined Exceptions Due: November 3rd, 11:59 pm

**Objectives: The focus of this assignment is to implement the LinkedList data structure.**

**Program Description:**

This project will alter the EmployeeManager adding LinkedLists for each of the individual type of Employees to make certain operations quicker. The LinkedList elements will simply refer to the same objects as the ones in the ArrayList. When Employees are added/removed from the ArrayList they must also be removed from the correct LinkedList.

A total of thirteen classes are required.

ListNode – Node used for LinkedList

LikedList – Implementation of the LinkedList data structure of Comparables

EmptyListException – Exception thrown by some LinkedList methods when performed on an empty list

ArrayList (From previous assignment)

InvalidEmployeeNumberException (From previous assignment)

InvalidSizeException (From previous assignment)

MaximumCapacityException (From previous assignment)

Employee (From previous assignment)

HourlyEmployee (From previous assignment)

SalaryEmployee (From previous assignment)

CommissionEmployee (From previous assignment)

EmployeeManager (Altered from previous assignment)

EmployeeDriver (From previous assignment)

It is up to you to implement ListNode, LinkedList, and EmptyListException. For the ListNode and LinkedList some of the code was reviewed in class and a non-Generic version for those methods are in the book.

**UML DIAGRAM FOR AND DISCUSSION FOR EmptyListException**

|  |
| --- |
| EmptyListException extends RuntimeException |
|  |
| <<constructor>>EmptyListException( )  <<constructor>>EmptyListException(name : String) |

Constructors

The constructor that takes a String as an argument calls upon the super class constructor with that String concatenated with “ is Empty”. The constructor that takes no argument calls upon the other constructor with the argument of “List”.

This class should belong to the exceptions package.

**UML DIAGRAM FOR AND DISCUSSION FOR ListNode**

|  |
| --- |
| ListNode<E extends Comparable<E>> |
| * data : E * nextNode: ListNode<E> |
| <<constructor>>ListNode(d : E)  <<constructor>>ListNode(d : E, node : ListNode<E>)  + setData(d : E)  +getData() : E  +setNext(next : ListNode<E>)  +getNext() : ListNode<E> |

Notes on ListNode

ListNode(d : E) sets the nextNode to null, the rest of the implementation of the ListNode class is self-explanatory as discussed in class

**UML DIAGRAM FOR AND DISCUSSION FOR LinkedList**

|  |
| --- |
| LinkedList<E extends Comparable<E>> |
| * firstNode : ListNode<E> * lastNode : ListNode<E> * numElements : int * name : String |
| <<constructor>>LinkedList()  <<constructor>>LinkedList(name : String)  + insertAtFront(item : E)  + insertAtBack(item : E)  + removeFromFront() : E throws EmptyListException  + removeFromBack() : E throws EmptyListException  + removeItem(index : int) : E throws IndexOutOfBoundsException  + getItem(index : int) : E throws IndexOutOfBoundsException  + setItem(index : int, item : E) throws IndexOutOfBoundsException  + findAndRemove(item : E) : Boolean  + findItem(item E) : int  + lengthIs() : int  + clear()  + toString()  + isEmpty() : Boolean  + sort() throws EmptyListException |

Notes on LinkedList

Constructors

Both constructors set firstNode and lastNode to null and numElements to 0. The constructor that takes a String sets the name data member to the String that is passed

*insertAtFront(E)*

Inserts the passed item to the front of the list

*void insertAtBack(E)*

Inserts the passed item in the back of the list

*void removeFromFront()*

Removes the first item in the list, and returns it. If the list is empty throws a new EmptyListException with the message: “(Name of list) is Empty”

*removeFromBack()*

Removes the last item in the list, and returns it. If the list is empty throws a new EmptyListException with the message: “(Name of list) is Empty”

*removeItem(int)*

Removes the element from the list at the given index. If that index does not exist within the LinkedList an IndexOutOfBoundsException is thrown with the message: “(Name of List) Index out of Range”. Returns the item removed.

*getItem(int)*

Returns the element at the given index. If that index does not exist within the LinkedList an IndexOutOfBoundsException is thrown with the message: “(Name of List) Index out of Range”.

public setItem(int index, E item)

Attempts to place the passed item into the given index. If that index does not exist within the LinkedList an IndexOutOfBoundsException is thrown with the message: “(Name of List) Index out of Range”.

*findAndRemove(E)*

Attempts to find the passed item, if found removes it and returns true, if not returns false

*findItem(E)*

Attempts to find the passed item, if found returns the location, if not returns -1

*lengthIs()*

Returns the number of elements in the list

*clear()*

Removes all elements from the list

public String toString()

Returns a String containing all elements in the LinkedList separated by two new lines.

public void sort()

Sorts the contents of the LinkedList using the Selection Sort. If the list is empty throws a new EmptyListException with the message: “(Name of list) is Empty”

*isEmpty()*

Returns true if empty, false if not

Both the ListNode and LinkedList class belong to the dataStructures package.

Changes to EmployeeManager

The EmployeeManager is being updated to use LinkedLists to reference the Employees in the ArrayList by sub-type. Changes need to be made where Employees are add/removed to perform similar actions to the correct LinkedList. Additionally some of the previous methods can be changed to take advantage of this addition. Since this now uses your LinkedList you must import it from your dataStructures package (changes in bold).

|  |
| --- |
| EmployeeManager |
| - employees : ArrayList<Employee>  - employeeMax : final int = 10  **- hourlyList : LinkedList<Employee>**  **- salaryList : LinkedList<Employee>**  **- commissionList : LinkedList<Employee>** |
| **<<constructor>> EmployeeManager()**  **+ addEmployee( type : int, fn : String, ln : String, m : char, g : char, en : int, ft : boolean, amount : double) throws InvalidEmployeeNumberException**  **+ removeEmployee( index : int)**  + listAll()  **+ listHourly()**  **+ listSalary()**  **+ listCommision()**  + resetWeek()  + calculatePayout() : double  + getIndex( empNum : int ) : int  + annualRaises()  + holidayBonuses() : double  + increaseHours( index : int, amount : double)  + increaseSales( index : int, amount : double)  + findAllBySubstring(find : String) : Employee[]  - RabinKarp(name : String, find : String) : int  - stringHash(s : String) : int  - charNumericValue(c : char) : int  - RabinKarpHashes(s : String, hashes : int[], pos : int, length : int) : int  - linearSearchRecursive(nameHashes : int[], findHash : int, pos : int) : int  **+ sort()** |

Constructor

Create LinkedLists for the new data members. Name the lists (by using the appropriate constructor) “Hourly List”, “Salary List”, and “Commission List”.

addEmployee

If adding the Employee to the ArrayList is successful, add the same object to the appropriate LinkedList depending on the sub-type of that Employee.

removeEmployee

When removing an Employee from the ArrayList also remove it from the appropriate LinkedList depending on the sup-type of that Employee.

listHourly, listSalary, listCommission

Alter these to output the contents of the corresponding LinkedList.

Continued on next page ->

sort

Also sort the LinkedLists

**Other Notes:**

* Declare the new classes as belonging to the correct packages
* Import the new classes as necessary.
* The functionality when tested through the EmployeeDriver will be identical to the previous iteration of the project
* Since the usage of the LinkedList in the EmployeeManager does not require all LinkedList methods I will also be testing the LinkedList itself in a separate program, I will post the test program for you to use, the expected output of the test program is shown below

# **Submitting Your Program Files:**

The assignment is automatically collected from your account on the “Loki” machine on the due date. You must put your source code in the correct directory in your account in order for this to work. Your directory name includes the course, section, semester and the assignment number. This is case-sensitive; **all alphas are upper-case**. This directory must be created in your home directory.

Submission Directory:

**CSCI-1620-1-F15-A5**

Running the LinkedListTest class should produce the following output ***EXACTLY***, in both values and spacing. If your output doesn’t look exactly like this, you will not get full credit:

Length is: 0

Populating from front

Length is: 5

4

3

2

1

0

Populating from back

Length is: 10

4

3

2

1

0

0

1

2

3

4

Sorting

Length is: 10

0

0

1

1

2

2

3

3

4

4

Removing from front, then back

Length is: 8

0

1

1

2

2

3

3

4

5 not removed

Length is: 8

0

1

1

2

2

3

3

4

0 removed

Length is: 7

1

1

2

2

3

3

4

Value of 5 not found

Value of 2 found at 2

Value at position 2 being removed

Length is: 6

1

1

2

3

3

4

Removing element 0

Length is: 5

1

2

3

3

4

Removing from front

Length is: 4

2

3

3

4

Getting value at position 2

Value at postion is 3

Setting position 0 and 3 to 5

Length is: 4

5

3

3

5

Testing IndexOutOfBoundsException

java.lang.IndexOutOfBoundsException: Integer List Index out of range

java.lang.IndexOutOfBoundsException: Integer List Index out of range

java.lang.IndexOutOfBoundsException: Integer List Index out of range

java.lang.IndexOutOfBoundsException: Integer List Index out of range

java.lang.IndexOutOfBoundsException: Integer List Index out of Range

java.lang.IndexOutOfBoundsException: Integer List Index out of Range

End of IOBE test

Clearing if not empty

Length is: 0

Testing EmptyListException

exceptions.EmptyListException: Integer List is empty

exceptions.EmptyListException: Integer List is empty

exceptions.EmptyListException: Integer List is empty

End of EmptyListException test