**CS 1450 Data Structures and Algorithms – Spring 2020**

**Assignment #9**

Due Date: **April 22, 2020** at 1:40pm (MW class), **April 23, 2020** at 1:40pm (TR class)

Purpose: Learning to create a linked list.

Effort: Individual

Points: **100**

Deliverables: Upload a **.zip** file with your source code (.java file) and design notes to Canvas by due date. **Hand in your design notes by taking photos or scanning.** Be sure to put your name on your design notes, then select all files and place into ONE zip file.

**Assignment Description**

This assignment provides the opportunity to implement a singly-linked list with a variety of methods. The linked list for this assignment stores elephants and performs tasks to add elephants into the linked list, find the total weight of all the elephants, then remove the elephants one at time from heaviest to lightest.

The completed linked list program contains the following classes:

* 1 Elephant
* 1 Linked List
* 1 Node (inner class of the Linked List)

**Specifications**

1. Create a Java class called **UsernameAssignment9**
2. Follow "CS1450 Programming Assignments Policy"
3. Write a test program (i.e. main) to do the following:
   1. Create an instance of the linked list class: **ElephantLinkedList**
   2. Step 1: Create elephants from *elephants.txt* and add them to the linked list
      1. For each elephant in the file:
         * Read the information for the elephant
         * Create an elephant object
         * Add the elephant object to the end of the linked list
           1. Use the ***add*** method in the ElephantLinkedList
      2. **Print the linked list after all elephants are added**
         * Print the elephant’s name and weight.
         * Use the ***printList*** method in ElephantLinkedList
   3. Step 2: Find the total weight of all elephants in the linked list
      1. Traverse linked list to compute the total weight – ***getTotalWeight*** method
      2. **Print the total weight of all elephants in the linked list**
   4. Step 3: Remove all elephants from the linked list, from heaviest to lightest
      1. Traverse linked list and find the elephant that weighs the most – ***findHeaviest*** method
      2. **Print the name and weight of the heaviest elephant**
      3. Traverse list again looking for that specific elephant and remove it – ***remove*** method
      4. Repeat until all elephants have been removed.
      5. Draw sketches showing pointers at the different stages to help solve.

**Classes**

**Elephant**

* Description
  + Class that represents one elephant.
* Private Data Fields
  + **name** – string for the name of the elephant
  + **weight** – integer for weight of the elephant
* Public Methods
  + Constructor
    - Initialize name and weight to incoming values for name and weight
  + Getters: for data fields name and weight
  + Setters: none
  + **public** method ***print()***
    - Prints the name and weight of the elephant in a nice format

**ElephantLinkedList Class**

* Description
  + A singly linked list constructed from nodes containing elephants.
  + Each node contains one elephant.
* Private Data fields
  + **head** – reference to the 1st node in the linked list
  + **tail**– reference to the last node of the linked list
  + **Note: do not include a size variable for the list, write the code without knowing the number of nodes.**
* Public Methods
  + Constructor
    - Initializes head and tail references to null
  + ***public boolean isEmpty()***
    - Return true or false based on if the linked list is empty
  + ***public int getTotalWeight()***
    - Return the total weight of all elephants in the linked list
  + ***public void add (Elephant elephant)***
    - Add a new node with the elephant to the end of the linked list
  + ***public Elephant findHeaviest ()***
    - Find and return the elephant that weighs the most, if list is empty returns null
  + ***public void remove(Elephant elephant)***
    - Remove an elephant from the linked list.
  + ***public void printList()***
    - Display the name and weight of all elephants in the linked list
* Private Inner Class
  + Node
    - Description
      * Represents **one node** in the linked list the stores an elephant in the data field.
      * See more details about inner classes in the Tips section.
    - Private Data Fields
      * **elephant** – the data stored in a node will be an elephant object
      * **next** – reference to next node in linked list (note: data type is Node!)
    - Public Methods
      * Constructor
        + Initializes elephant to incoming elephant
        + Initializes next reference to null

**Test File Information**

The file **elephants. txt** contains one line for each elephant. Again, this is a test file, do NOT ASSUME the names, weights, order or number of elephants will be the same in the file used for grading.

All elephants have unique names and weight. The format of each line in the file is as follows:

name weight

**Nella 8293**

**Tips**

**Must Do: Create your own linked list**

* **DO NOT** use the **java.util.LinkedList<E>** class provided in Java Collections Framework
* Create a linked list class **(ElephantLinkedList)**
* Create a private inner Node class in your linked list **(Node)**
* Create the methods for the linked list (add, remove, etc.)

**Tip: Linked List Class**

* You **DO NOT** need to create a generic linked list or generic node as was done in the lectures.
* Your linked list class must contain an inner node class that stores elephants.

**Tip: Build the classes first**

* Before writing the code in main, build the classes from the details given above.
* Work on your code incrementally. Write the code in main that adds and prints the linked list to ensure your code is working, then write another method and test, etc.

**Tip: Inner Class**

* An inner class, or nested class, is a class defined WITHIN the scope of another class.
* Generally, we define a class as an inner class if it is used **ONLY** by its outer class.
* This is exactly the case for the Node class.
* Node is part of the linked list and will **only** be used by the linked list, that is why it should be a private inner class. It serves as the private data storage unit for a linked list.
* General **code layout** when including an inner class is something like the following:

**class** ElephantLinkedList {

// Private data fields go here

// Public methods: add, findHeaviest, printList, etc. go here

**private** **static** **class** Node {

The node class is **INSIDE** the

} // Node ElephantLinkedList class and

Specified as **private static**

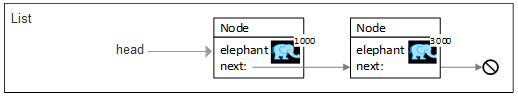
} // ElephantLinkedList

* **NOTE**
  + Because the Node class is private it is **hidden** from users of the linked list class.
  + The data fields of Node may be **accessed directly** within ElephantLinkedList
    - Accessing directly is when you use dot notation (no getters/setters are needed):
      * current = current.next;
      * current.elephant.getWeight()

Can access **elephant** directly since Need getter to access elephant’s **weight** since

current is reference to inner class weight is a **private** variable in external class

**List with two elephants**



**Output**

Step 1: Created and placed elephants in linked list

Elephant Weight

--------------------------------

Nella 8293

Libby 5922

Echo 10234

Titan 12523

Roxy 7042

Jumbo 5007

Izzy 10023

Step 2: Find total weight for all elephants

Total weight of all elephants is: 59044 lbs

Step 3: Find and remove elephants, heaviest to lightest

Removing: Titan weighing in at 12523 lbs

Removing: Echo weighing in at 10234 lbs

Removing: Izzy weighing in at 10023 lbs

Removing: Nella weighing in at 8293 lbs

Removing: Roxy weighing in at 7042 lbs

Removing: Libby weighing in at 5922 lbs

Removing: Jumbo weighing in at 5007 lbs