Ordered Collection of Things using Linked List

Goal: Can represent an ordered collection using a linked list

Problem Description

You will be creating an ordered collection class. You will be storing your things in a linked list.

- You may NOT use the Java library classes `ArrayList` or `Vector` or `LinkedList` or any other java collection class.
- You **must** implement your own linked list as explained below.
- No method in the collection class should print anything!!!!
- The name of your collection class should include the name of your Thing. For example, if your Thing is called `Book`, then your collection class should be called `BookCollection`. Alternatively, if your collection class has a natural group name, please use that. This will help with "thinking" in objects. For example, a collection of books can be called a library.
- Differentiate this class from the array implemented class by adding Linked in the class name.

ThingNode Class

Implement a node class for your linked list. Name it after your Thing (for example: McDonaldsNode). Your node class should include the following:

1. Two private instance variables:
    - data of type "Thing". Do NOT use a generic node.
    - link of type "ThingNode".
2. A constructor that takes two input parameters and uses them to initialize the two instance variables.
3. Getters and Setters for the instance variables.

LinkedCollection Class

Implement a collection class of your things using a linked list. Your collection class must have the following characteristics:

1. Include exactly one instance variable:
    - head of type ThingNode
2. Implement a no arguments constructor for your collection class.
3. **`void insert(Thing):`** a method that takes one input parameter matching the type of your Thing and then inserts it in the collection class IN DESCENDING ORDER according to the search key instance variable. Simply add any duplicate Things. Do NOT attempt to sort the list or use a sorting algorithm. You must maintain the list in order by inserting the new element in the proper position.
4. **`void delete(Thing):`** this method takes one input parameter of your type of Thing. The method then searches the collection for the first object that equals the input object and deletes its occurrence if found. After the item has been deleted, the list must still be maintained IN ORDER.

5. **`int size():`** a method that returns the number of objects in the collection.

6. **`boolean isEmpty() :`** a method that returns true if there are no objects in the collection, false otherwise.

7. **`int indexOf(Thing):`** a method that returns the index of the first instance of Thing found in the collection. Returns -1 if not found. Recall that numbering starts at one for linked lists.

8. **`int lastIndexOf(Thing):`** a method that returns the index of the last instance of Thing found in the collection. Returns -1 if not found.

9. **`Thing grab(int):`** a method that returns the Thing located at the specified position in the list or null if position is not valid. Recall that the first element in the list is at position 1. This method does not remove the element from the list.

10. **`boolean contains(Thing):`** this method takes one input parameter matching the type of your Thing. The method returns `true` or `false` based on whether the collection contains at least one Thing that is equal to the input parameter. Use the `equals()` method from your Thing class to determine if two objects are equal.

11. **`String toString():`** a method that returns a String representation of the collection that includes all elements that are stored in the collection. The output string must be nicely formatted in a tabular format, one Thing per line. toString() will not print but rather will return a String. For example, a String representing a list of students (in reverse order by name) could be formatted as follows:

```
Name          Age         Registered
---------------------------------
Reem          27          true
Mohammed      25          false
Maria         20          true
Hanna         30          true
```

12. **`int countOccurrences(Thins):`** this method returns a count of the number of Things in the collection that are equal to the input parameter.

13. Implement the Iterable interface and the iterator() method for your collection class. iterator() returns an instance of an embedded iterator class.

    The iterator class will have the following methods:

    - constructor
    - hasNext() : returns true if there is another item in the collection, false otherwise
    - next(): returns the next item in the collection

    The iterator should return the items in reverse alphabetical order.

Extra methods for more practice

14. **`int total():`** this method uses the `int` instance variable of your Thing class. The method calculates and returns the sum of the `int` instance variables of all objects stored in the collection.

15. **`Thing find(String):`** this method depends on the search key attribute of your Thing class. The method takes as input a `String` value and returns the first object with a search key that matches the input search value.

16. **`double average():`** this method returns the average of the `int` instance variables of all objects stored in the collection.

17. **`int countRange(int low, int high):`** this method depends on the `int` instance variable of your Thing class. The method takes as input two `int` values and counts how many objects in the list have a value that lies in the given range including the endpoints. Note that if the first input parameter is greater than the second input parameter then the method always returns 0 (i.e., `countRange(30,25)` returns 0).

Driver Class

Create a driver by copying the driver from your array project. Modify the line where you create the instance of the collection class (to be an instance of the linked collection instead of the array collection). Does anything else need to be modified to create the following functionality??

- Create an instance of your collection class
- Add 5 – 10 "Things" to your collection class
- Verify that all methods work as expected