CSCI 3302 Programming Assignment 03 (100 Points)

Due: Oct 10, 12:00 AM

OBJECTIVES:

- Demonstrate basic competency in using linked list concepts.
- Implement a Node-based Set ADT.

ASSIGNMENT ASSISTANCE:

- This homework assignment is due at the time listed above.
- This assignment is restricted to individual effort.
- As per our syllabus, the use of AI is prohibited. Any resource used (other than Dr. Becnel or the course text) must be documented in the code (as comments) detailing the source and describing exactly what was learned and how that information was used. Submissions will be severely penalized if copied in part or whole from any source.
- If you need help, visit your instructor during his posted office hours. If your schedule cannot accommodate any of these times, email your instructor to schedule a different time.

PROBLEM DESCRIPTION:

1. The starter code has the following files:
   a. SetADT.java – interface for the abstract data type set. This file shall NOT be changed.
   b. Node.java – standard Node class that we have used several times. This file shall NOT be changed.
   c. Set.java – set class with a `toString`() and `equals`() method already defined (but not working until you implement the SetADT interface).

2. Your Set implementation should use generics and have one class attribute, of type `Node<E>` called `leadNode`. This node holds an element in the set and have a link to another element.

3. You will create two public constructors:
   a. One constructor takes no arguments and create an empty set, i.e. a set with no elements
   b. The second constructor will take one argument called `element` of Generic type (E) and create a set with one element by creating the `leadNode` with the given element.

4. Each method in SetADT has a description. The following should be straightforward:
   a. `isEmpty`
   b. `size`
   c. `removeAll`

   Note: you are **NOT** allowed to create another class attribute/field, such as numElements.

5. The `contains` method is fairly straightforward. Here are some examples:
   a. {2, 3, 4} contains 2 – true
   b. {2, 3, 4} contains 7 – false
   c. { } contains 4 – false; Here { } is an empty set with nothing in it.

6. The `isSubsetOf` method determines if the implicit set is a subset of the set given in the parameter. For a set A to be a subset of a set B, we must have all elements of A appear in B. Here are some examples:
    a. {2, 3} isSubsetOf {2, 3, 4} – true
    b. {2, 3} isSubsetOf {2, 3} – true
    c. {2, 3} isSubsetOf {4,3,2,1} – true; Note: the order of elements in a set does not matter
    d. { } isSubsetOf {2, 3, 4} – true; Here { } is the empty set and has no elements; so vacuously all elements appear in {2, 3, 4}.
    e. {2, 3, 4} isSubsetOf {2, 4} – false; Here 3 does not appear in {2, 4}, {2, 3, 4} is not a subset of {2, 4}.

   Note: The equals method (already in the file) uses isSubsetOf. The order does not matter, only the elements. So, {2,3} and {3,2} are the same set, because they have the same elements.

7. The `add` method takes the given element and puts the element in the set. Since the order does not matter, it is easiest just to put the element in the front of the set. That is, make a new `leadNode` and put the element in this new `leadNode`. <u>Note: we do NOT allow duplicate elements in sets.</u> Here are some examples:
    a. add 2 to { } – results in {2}
    b. add 3 to {2} – results in {3, 2}
    c. add 4 to {3,2} – results in {4, 3, 2}
    d. add 2 to {4, 3, 2} – results in {4, 3, 2}. Note: the set is unchanged.

8. The `union` method creates a new set that has all the elements in <u>either</u> of the given sets (the implicit set `this` and the given set argument). Here are some examples:
    a. {a} union {b} – results in {a,b}
    b. {a, b} union {c, d} – results in {a, b, c, d}
    c. {a, b} union {a, b, c} – results in {a, b, c}; Note: we do not duplicate elements.
    d. { } union { } – results in { }
    e. { } union {a, b, c} – results in {a, b, c}

   A new set is created in this method and returned.

9. The `intersect` method creates a new set that has all the elements found in <u>both</u> of the given sets (the implicit set `this` and the given set argument). Here are some examples:
    a. {a, b, c} intersect {z, a, b, y} – results in {a,b}
    b. {a, b} intersect {c, d} – results in { }
    c. {a, b} intersect { } – results in { }

   A new set is created in this method and returned.

10. The `remove` method takes a given element and removes the element from the set. If the set does not contain the element, then nothing happens:
    a. remove 2 from { } – results in { }
    b. remove 2 from {1, 3, 4} – results in {1, 3, 4}
    c. remove 2 from {2, 3, 4} – results in {3, 4}
    d. remove 4 to {2, 3, 4} – results in {2, 3}

SUBMISSION:

- Review the Evaluation below to ensure you have met all the requirements.
- Commit `Set.java, Node.java,` and `SetADT.java` to GitHub. You can include other files but they will not be graded. Upload a backup copy to D2L.

EVALUATION

Remember to consult Program Requirements.docx

Automatic Deductions:	
Late/Not Submitted	-100
Code not submitted to GitHub	-30
Code does not run/compile	-50
Node.java or SetADT changed	-50
Earn Points for the following:	
Code has a comment header with name, section, date	5 pts
Code organization, structure, and indention are appropriate (SHFT + ALT + F in VS Code)	5 pts
Code is well and meaningfully commented.	5 pts
Fields and Constructor correct	10 pts
isEmpty and removeAll correct	10 pts
size method	5 pts
contains method	10 pts
isSubsetOf of method	10 pts
add method	10 pts
union method	10 pts
intersect method	10 pts
remove method	10 pts