University of Bahrain

College of Information Technology

Department of Computer Science

First Semester, 2018-2019

**ITCS214 (Data Structures)**

**Tutorial 3**

**Question 1 (A)** Write a generic class called **SingleLinkedList**having following data fields (private):

private Node<E> head; // reference to the first node

private int size; // number of nodes

This class is also having an inner class called **Node** (as discussed in the lectures).

The class **SingleLinkedList** will have the following private methods:

|  |  |
| --- | --- |
| **Private Method** | **Behavior** |
| private void addFirst (E item) | Inserts a new node in the beginning of the list with item as *data* |
| private void addAfter (Node<E> nodeRef, E item) | Inserts a new node with item as data after the node with reference *nodeRef* |
| private E removeAfter (Node<E> nodeRef) | Deletes the node after the node referenced by *nodeRef* and returns the *data* of the node deleted |
| private E removeFirst () | Deletes the first node of the list and returns the *data* of the node deleted |
| private Node<E> getNode(int index) | Returns reference to the node at position given by *index* |

The class **SingleLinkedList** will have the following public methods:

|  |  |
| --- | --- |
| **Public Method** | **Behavior** |
| public SingleLinkedList ( ) | Constructor to initialize *head* to null and *size* to 0 |
| **public int size()** | Returns current *size* |
| **public boolean contains(E obj)** | Checks whether the given object *obj* is present in the list. If it is there then it returns true else it returns false. |
| **public void clear()** | Removes all the elements of the linked list and makes it empty |
| public boolean isEmpty() | Checks whether the list is empty or not |
| public boolean add ( E anEntry) | Adds object *anEntry* at the end of the list and returns true. |
| public void add (int index, E anEntry) | Adds object *anEntry* in the list at the location given by *index* |
| public E get (int index) | Returns the element of the list at position given by *index* |
| public E set (int index, E newValue) | Updates the element at position index by *newValue* and returns the old value |
| public E remove (int index) | Removes the element at position *index* and returns the element being removed |
| public boolean remove (E obj) | Removes the first occurrence of the object *obj* from the list, if present and returns true, else returns false. |
| public int indexOf(E obj) | Returns the *index* of the first occurrence of the specified element obj in this list, or -1 if this list does not contain the element. |
| public String toString() | Returns the String equivalent of the list object |

**(B)** Write a class called SLListTest having only main method to test all

functionalities.

**Question 2** Rewrite the following methods of class **SingleLinkedList** without calling any other method:

1. public void add (int index, E item)
2. public boolean remove (E item)

**Question 3** Write following methods to be included in class **SingleLinkedList** of **Question 1**. You may call other methods of the class **SingleLinkedList** in your method.

1. **removeAll**: Removes all occurrences of the object **obj** from the list, if present and returns true, else returns false.

Method heading: **public boolean removeAll(E obj);**

1. **equals**: Returns true if “this list” and parameter **list** contains the same elements in the same order, else returns false.

Method heading: **public boolean equals(SingleLinkedList<E> list);**

1. **Copy constructor:** creates a copy of the list object of type **SingleLinkedList.**

Method heading: **public SingleLinkedList(SingleLinkedList<E> list);**

**(D) reverse**: The method reverses the elements of the linked list. If the list is empty or has only one node return false, otherwise return true, after reversing the list.

Method heading: **public boolean reverse()**

Example:

Before Calling list: 10 20 30 55 60 70 80

After Calling list: 80 70 60 55 30 20 10

**Question 4** Write a method **compareHalfs** to be included in the **SingleLinkedList** class that does not accept any parameter. The method returns true, if all the elements of the first half of the linked list are greater than all the elements of the second half of the list, otherwise it returns false. If the list is empty or does not have even number of nodes, the method returns false.

Hint: Find the minimum of the first half of the linked list and the maximum of the second half of the linked list and compare. Use **compareTo** method of interface **Comparable** to compare data items.