**CS 315: Reference Variables and Linked Lists**

|  |  |  |
| --- | --- | --- |
|  | Be careful when working with reference variables: |  |
|  |  | What happens to 5??? |

**Implementing an Indexed List using Linked Lists**

|  |  |
| --- | --- |
| package List;  class Node {  Object item; // item in list  Node next; // link to the next node  Node(Object newItem) {  item = newItem;  next = null;  } // end constructor  Node(Object newItem, Node nextNode) {  item = newItem;  next = nextNode;  } // end constructor  } // end class Node  //p. 251 of text |  |

|  |  |
| --- | --- |
| **Exercise**: Draw an appropriate picture showing what would happen if the following code were executed?  Node n = new Node (new Integer(6));  Node first = new Node (new Integer(9), n); |  |

|  |  |
| --- | --- |
| NOTE: In ListReferencedBased:   * We want the data field next in the last node to be **set to null** * **head reference variable** * References the list’s first node * Always exists even when the list is empty | public class ListReferenceBased implements ListInterface {  // reference to linked list of items  private **Node head**;  private int numItems; // number of items in list  public ListReferenceBased() {  numItems = 0;  head = null;  } // end default constructor |

**Exercises:** Given the Code to implement List ADT as an Array, Write implementation Code to implement the List ADT as a Linked List:

|  |  |
| --- | --- |
| // Code to implement List ADT as an Array  public boolean isEmpty() {  return (numItems == 0);    } // end isEmpty | // Code to implement List as a Linked List  public boolean isEmpty() {    } // end isEmpty |
| public int size() {  return numItems;    } // end size | public int size() {  } // end size |
| public void removeAll() {  // Creates a new array; marks old array for garbage collection.  items = new Object[MAX\_LIST];  numItems = 0;  } // end removeAll | public void removeAll( ) {  } // end removeAll |

|  |  |
| --- | --- |
| 1) | aList.head  **Before:**    **AFTER the client code: aList.removeAll();** |

Exercises: Give the following indexed list, aList that is implemented as a linked list, draw a picture of the linked list AFTER each of the following client code.

|  |  |
| --- | --- |
| 2. | aList.head  **Before:**    **AFTER the client code: aList.add(2,50):** |

3)

|  |  |
| --- | --- |
| **Before** | After the client code: aList.add(2, 30); |

Head

4. Given the following linked list:



If possible, draw a picture to show what the linked list (named NumberList) would look like if the following independent Java Client code were executed.

|  |  |
| --- | --- |
| Client Code | Picture |
| a) NumberList.add(0, 0); |  |

|  |  |
| --- | --- |
| b) NumberList.add(4, 4); |  |
| c) NumberList.add(5,10); |  |
| d) NumberList.add(2,7); |  |

5. In the referenced-based implementation of the ADT list, there is also a private method**, find** that is used in the ***add,*** ***get*** and ***remove*** methods. Look at the picture, and then write the code for this method.

|  |
| --- |
| **Before:**  aList.head |
|  |
| **After:** Implementation code: find(2) returns: |

|  |  |
| --- | --- |
| private Node find(int index) {  // --------------------------------------------------  // Locates a specified node in a linked list.  // Precondition: index is the number of the desired  // node. Assumes that 1 <= index <= numItems+1  // Postcondition: Returns a reference to the desired  // node.  // --------------------------------------------------    } // end find | // Code:  private Node find(int index) {  // end find |

|  |  |
| --- | --- |
| 6) | **Before:**  aList.head    After the client code: aList.remove(1) is executed. |

|  |
| --- |
| 7) Before: After the client code aList.remove(0) |

8. Given the following linked list:

head



If possible, draw a picture to show what the linked list (named NumberList) would look like if the following independent Java client code were executed. Be sure you see what lines of code in the method would be executed

|  |  |
| --- | --- |
| Client Code | Picture of Implementation |
| a) NumberList.remove(0); |  |

Given the following linked list, draw a picture ( if possible) to show what the linked list (named NumberList) would look like if the following independent Java client code were executed. Be sure you see what lines of code in the method would be executed

head



|  |  |
| --- | --- |
| b) NumberList.remove(4); |  |
| c) NumberList.remove(2); |  |
| d) NumberList.remove(76); |  |

We will now write implementation code for**: get, remove** and the ***add*** methods. We will also write code for testing these methods.

|  |
| --- |
| **public Object get(int index) throws ListIndexOutOfBoundsException {**  **} // end get** |
| **public void add(int index, Object item) throws ListIndexOutOfBoundsException, ListException{**  **} //end add** |
| public void remove(int index) throws ListIndexOutOfBoundsException {  } // end remove |
| public class TestLinkedListCode  {  static public void main(String[] args) {  ListReferenceBased TestList = new ListReferenceBased();  … }} |