HDSE 222

Data Structures and Algorithms Practice

P3 – Lists implementation

By,
Dr. Tharinda Nishantha Vidanagama.
Senior Lecturer,
CMIS.

Java code for list item

- The list items only contain and int data item

```
public class link {

    public int iData; // data item
    public link next; // next link in list

    public link(int id) {// constructor
        iData = id; // initialize data
        // ('next' is automatically set to null)
    }

    public void displayLink() { // display ourself
        System.out.print("{" + iData + "} ");
    }
} // end class Link
```

Java Linked list

- The list only contains a reference to the first node of the list.

```
public class linkedList {
    private link first; // ref to first link on list

    public void LinkList() { // constructor
        first = null; // no items on list yet
    }

    public boolean isEmpty() { // true if list is empty
        return (first == null);
    }

//....More methods
}
```

Java- insertFirst method

```
public void insertFirst(int id) {// insert at start of list
    link newLink = new link(id);// make new link
    newLink.next = first; // newLink --> old first
    first = newLink; // first --> newLink
}
```

- It works for empty list also!

Java - deleteFirst

```
public link deleteFirst() { // delete first item
    if (isEmpty())
        System.out.println("Empty List!!");
    else {
        link temp = first; // save reference to link
        first = first.next; // delete it: first-->old next
        return temp; // return deleted link
    }
    return null; //for empty list
}
```

- The statement “first = first.next” is all you need to remove the first link from the list.
- The link is also returned for the any other operations required on the linked list.
- It is saved in “temp” before deleting it, and then return the reference of “temp”.

Java – displayList

- Start at “first” and follow the chain of references from link to link.
- “current” points to (or technically *refers to) each link in turn. It starts off* pointing to first, which holds a reference to the first link.
- “current = current.next;” changes current to point to the next link, because that's what's in the next field in each link.

```
public void displayList() {
    System.out.print("List (first-->last): ");
    link current = first; // start at beginning of list

    while (current != null) { // until end of list,
        current.displayLink(); // print data
        current = current.next; // move to next link
    }
}
```

Sample application class

- Try adding, displaying and deleting some elements.

```
class linkedListApp {

    public static void main(String[] args) {

        linkedList theList = new linkedList(); // make new list

        theList.insertFirst(22); // insert four items
        theList.insertFirst(44);
        theList.insertFirst(66);
        theList.insertFirst(88);

        theList.displayList(); // display list

        while (!theList.isEmpty()) { // until it's empty,
            link aLink = theList.deleteFirst(); // delete link
            System.out.print("Deleted "); // display it
            aLink.displayLink();
            System.out.println("");
        }
        theList.displayList(); // display list
    } // end main()
} // end class LinkListApp
```

Java- find(key)

- Given a specific data key it finds the link.

```
public link find(int key) { // find link with given key
    if (isEmpty())
        System.out.println("Empty List!!");
    else {
        link current = first; // start at 'first'

        while (current.iData != key) {// while no match,
            if (current.next == null) // if end of list,
                return null; // didn't find it
            else
                // not end of list
                current = current.next; // go to next link
        }
        return current; // found it
    }
    return null; //for empty list
}
```

Java – delete(key)

- First, the method finds the element and then deletes the link.

```
public link delete(int key) {// delete link with given key
    if (isEmpty())
        System.out.println("Empty List!!");
    else {
        link current = first; // search for link
        link previous = first;

        while (current.iData != key){
            if (current.next == null)
                return null; // didn't find it
            else {
                previous = current; // go to next link
                current = current.next;
            }
        } // found it

        if (current == first) // if first link,
            first = first.next; // change first
        else // otherwise
            previous.next = current.next; // bypass it
        return current;
    }
    return null;// for empty list
}
```

Addition to App class

- Add these lines to the “linedListApp.java” class to execute the new methods.

```
link f = theList.find(44); // find item
if (f != null)
    System.out.println("Found link with key " + f.iData);
else
    System.out.println("Can't find link");

link d = theList.delete(66); // delete item

if (d != null)
    System.out.println("Deleted link with key " + d.iData);
else
    System.out.println("Can't delete link");
```

2. Double-ended linked lists

- Two references are required to keep track of first and last elements.

```
public class doubleLinkedList {
    private link first; // ref to first link
    private link last; // ref to last link

    public doubleLinkedList() { // constructor
        first = null; // no links on list yet
        last = null;
    }

    public boolean isEmpty() { // true if no links
        return (first == null);
    }
```

Java - insertFirst

- Create a new link with a constructor
- Only if the list is empty the “last” reference is also set to the new link.
- New link is set to the “first” in any case.

```
public void insertFirst(int dd) {// insert at front of list
     link newLink = new link(dd); // make new link
     if (isEmpty()) // if empty list,
          last = newLink; // newLink <-- last

     newLink.next = first; // newLink --> old first
     first = newLink; // first --> newLink
}
```

Java - insertLast

- Create a new link with a constructor
- Only if the list is empty the “first” reference is also set to the new link.
- New link is set to the “last” otherwise.

```
public void insertLast(int dd) { // insert at end of list
     link newLink = new link(dd); // make new link
     if (isEmpty()) // if empty list,
          first = newLink; // first --> newLink
     else
          last.next = newLink; // old last --> newLink
     last = newLink; // newLink <-- last
}
```

Java - deleteFirst

- Return null when list is empty
- Check whether there is only one element in the list.
- Set the first reference to the next element.

```
public link deleteFirst() { // delete first link
    if (isEmpty())
        return null; //empty list return null

    link temp = first; // save the data

    if (first.next == null) // if only one item
        last = null; // null <-- last

    first = first.next; // first --> old next
    return temp;
}
```

Java- remove last

- Check for empty list
- Check for one element
- Otherwise search for the 2nd last element.
- Set the second last element as the last.
- Return is just for displaying if required.

```
public link deleteLast() {// delete last link
    link temp = last; // keep data
    if (isEmpty())
        return null; // empty list return null

    if (first == last) // only one item in list
        first = last = null;
    else {
        link current = first;//start from first
        while (current.next != last)//find 2nd last link
            current = current.next;

        last = current;//change last
        current.next = null;//disconnect from last
    }
    return temp;
}
```

Sample double linkedListApp

- Insert some items to the list.

```
public class doubleLinkedListApp {

    public static void main(String[] args) {

        doubleLinkedList theList = new doubleLinkedList(); // make new list

        theList.insertFirst(22); // insert four items
        theList.insertLast(44);
        theList.insertFirst(66);
        theList.insertLast(88);

        theList.displayList(); // display list
```

Sample double linkedListApp (ii)

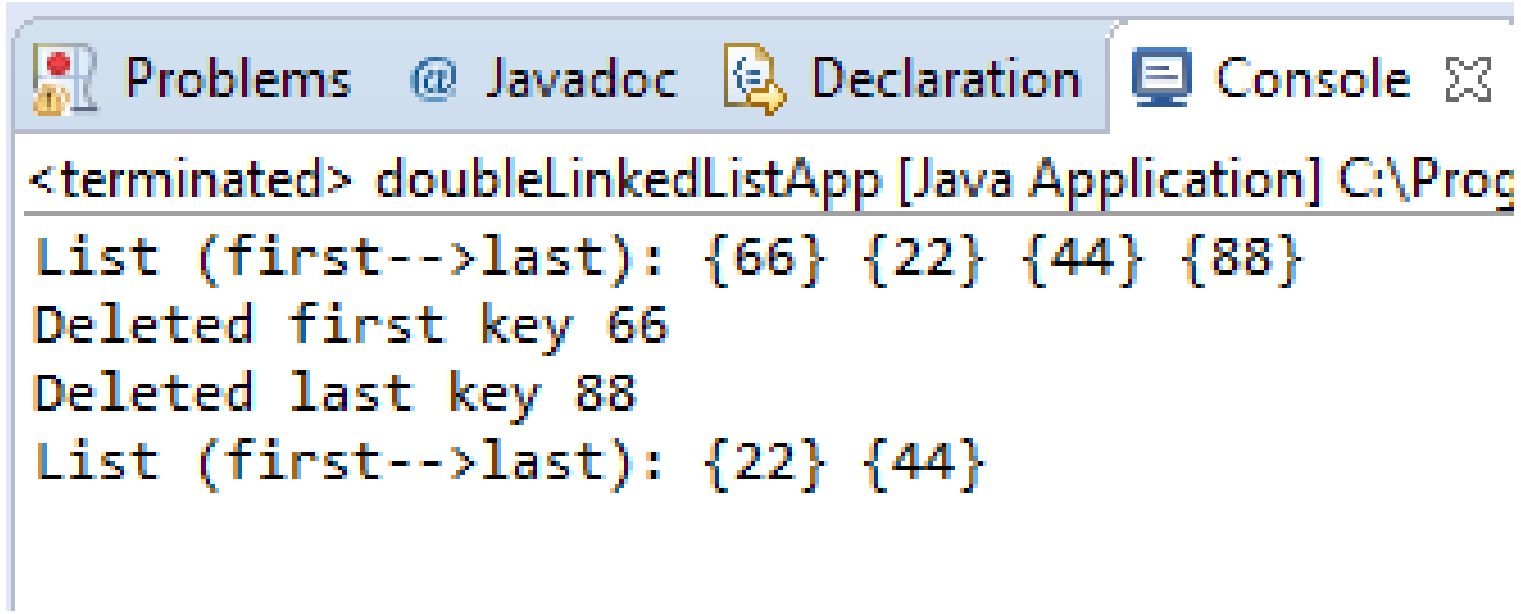
- Delete the first/ last nodes and display the list

```
        link f = theList.deleteFirst(); // delete first item
        if (f != null)
            System.out.println("Deleted first key " + f.iData);
        else
            System.out.println("Can't find link");

        link d = theList.deleteLast(); // delete last item
        if (d != null)
            System.out.println("Deleted last key " + d.iData);
        else
            System.out.println("Can't delete link");

        theList.displayList(); // display list
    }
}
```

Sample output



Java – insertMiddle

- Insert criteria may vary according to the application
- Declare two references to track the list.
- Find the preferred location
- Reassign the links to include the new node.

```
public void insertMiddle(int item, int afterKey){
    link newLink = new link(item); // make new link
    //assumes the list is not empty and
    // not inserting to first or last
    link current = first;//start from first
    while (current != last)
        if (current.iData != afterKey)//find key
            current = current.next;
        else
            break;
    newLink.next = current.next; // newLink
    current.next = newLink;
}
```

Java - deleteMiddle

- Assumes list is not empty and not deleting first or last
- Traverse the list until the required item is found.
- Rearrange the links to avoid referring to the deleting node.

```
public void deleteMiddle(int item){
    //assumes the list is not empty and
    // not deleting first or last
    link current = first;//start from first
    link prevCurrent = first;
    while (current != last)
        if (current.iData != item){//find key
            prevCurrent = current;
            current = current.next;
        }else
            break;
    prevCurrent.next = current.next; // delete current
}
```

Java - doubleEndedLinkedListApp

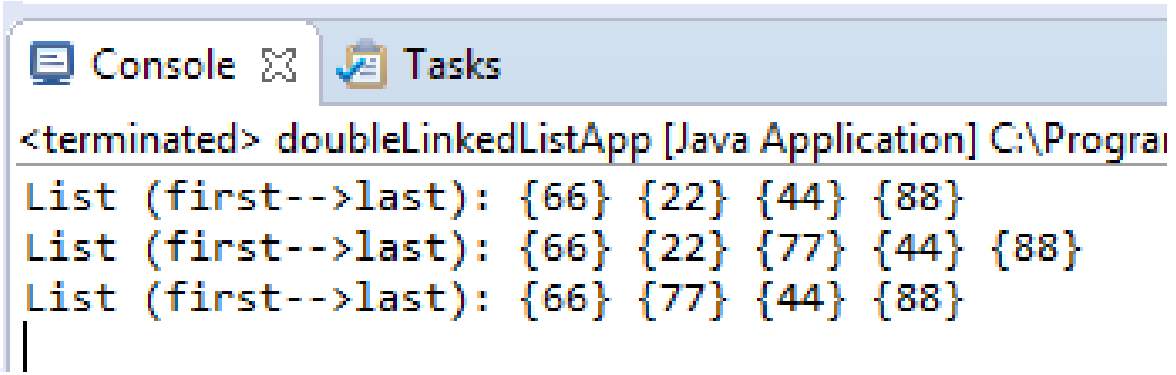
```
public static void main(String[] args) {

    doubleLinkedList theList = new doubleLinkedList(); // make new list

    theList.insertFirst(22); // insert four items
    theList.insertLast(44);
    theList.insertFirst(66);
    theList.insertLast(88);
    theList.displayList();

    theList.insertMiddle(77, 22);
    theList.displayList();

    theList.deleteMiddle(22);
    theList.displayList();
```



Assignment 1

- Implement the following methods
- int **count**() - return the number of elements in the Linked List.
- Object **remove**(int n) - remove the n^{th} element in the Linked List.
- void **add**(int n, Object item) - add the object item as the n^{th} element in the Linked List.
- Object **get**(int index) - return the element at the specified position in this list.

3. Doubly Linked Node

```
public class doublyNode {
    int data;
    doublyNode next;
    doublyNode previous;

    doublyNode(int x) {
        data = x;
        next = previous = null;
    }

    doublyNode(int x, doublyNode nextNode, doublyNode previousNode) {
        data = x;
        next = nextNode;
        previous = previousNode;
    }
```

```
    int getData() {
        return data;
    }

    doublyNode getNext() {
        return next;
    }

    doublyNode getPrevious() {
        return previous;
    }

}
```

Doubly Linked List class

- As before create head and tail link references.
- Constructor.
- Check for empty list.

```
public class DoublyLinkedList {
    private doublyNode head;
    private doublyNode tail;

    public DoublyLinkedList() {
        head = tail = null;
    }

    public boolean isEmpty() {
        return head == null;
    }
```

Doubly Linked List class (ii)

```
public void addToHead(int item) {
    if (isEmpty())
        head = tail = new doublyNode(item);
    else
        head = head.previous = new doublyNode(item, head, null);
}

public void addToTail(int item) {
    if (isEmpty())
        head = tail = new doublyNode(item);
    else
        tail = tail.next = new doublyNode(item, null, tail);
}
```

DLL – remove from head

- Check for empty list
- Keep data for returning
- If there is only one item need to handle specifically.

```
public int removeFromHead() {
    int item = 0;
    if (isEmpty())
        System.out.println("empty list!");
    item = head.data;
    if (head == tail)
        head = tail = null;
    else {
        head = head.next;
        head.previous = null;
    }
    return item;
}
```

DLL – Remove from tail

- Check for empty list
- Keep the data
- Special handling if only one item remaining.

```
public int removeFromTail() {
        int item = 0;
        if (isEmpty())
            System.out.println("empty list!");
        item = tail.data;
        if (head == tail)
            head = tail = null;
        else {
            tail = tail.previous;
            tail.next = null;
        }
        return item;
}
```

DLL – print from head to tail

```
public void print() {
    System.out.print("[ ");
    doublyNode current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.print("]\n");
}
```

DLL – print from tail to head

```
public void printReverse() {
    System.out.print("[ ");
    doublyNode current = tail;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.previous;
    }
    System.out.print("]\n");
}
```

DLL – sample application

```
public static void main(String[] args) {
    DoublyLinkedList dll = new DoublyLinkedList();

    dll.addToHead(33);
    dll.addToTail(55);
    dll.addToHead(22);
    dll.addToTail(66);

    dll.print();
    dll.printReverse();

    dll.removeFromHead();
    dll.print();

    dll.removeFromTail();
    dll.print();
}
```