

Static vs. Dynamic Data Structures

A *static data structure* is one whose size cannot change once it has been declared. In other words, the size of the data structure must be known at the time the data structure is created. An array is a good example of a static data structure.

A *dynamic data structure* is one whose size can change indefinitely once it has been declared. In other words, the size of the data structure does not need to be known at the time data structure is created. For the remainder of this course, we will focus on using and implementing dynamic data structures such as linked lists, stacks, queues, and binary trees.

Linked Lists

Sometimes a programmer does not know the maximum number of items that a program will need to store. In these cases, a static data structure such as an array cannot be used, because the maximum number of elements in an array is fixed while the program is executing.

A *linked list* is an example of a dynamic data structure. Linked lists can grow dynamically (while the program is executing) by dynamically allocating new objects via **new**.

A Linked List

Each *node* in the linked list contains some data and a reference (aka link) to the next node in the list. The link of the last node points to null.

In the list that follows, the data is just an integer.

3-> 7-> 5-> null

We keep two special pointers for each list:

- *head* refers to the first element of the list,
- *tail* refers to the last element of the list

Suppose we want to insert the integer 4 into the end of the above list. Then, how does the list look like?

Linked Node in Java

Before looking at the structure of a list, let's see the structure of an individual element of the list, which is called a *node*.

```
public class Node{
    private int number;
    private Node next; // refers to the next node

    // Constructor
    public Node(int n){
        this.number = n;
        this.next= null;
    }

    public void setNext(Node n) {this.next = n;}
    public int getNum() {return this.number;}
    public Node getNext() {return this.node;}
}
```

Linked List Class in Java

```
public class LinkedList{
```

```

private Node head, tail;

// Constructor
public LinkedList() {...}

// Methods that manipulate a list
public void insertHead(int n) {...}
public void insertTail(int n) {...}
public void printList() {...}
public void clear() {...}
public Boolean isEmpty() {...}
public int getHead() {...}
public void deleteHead() {...}
public void delete(int n) {...}
public LinkedList copy() {...}
}

```

Inserting into a Linked List

The following method adds a given integer *n* to the head of the list:

```

public void insertHead(int n)
{
    Node temp = new Node(n);
    if (this.head == null)
    {
        head = temp;
        tail = temp;
    }
    else
    {
        temp.setNext(this.head);
        this.head = temp;
    }
}

```

So accordingly, how can we implement an `insertTail` method in the `LinkedList` class?

More Linked List Methods

```

public LinkedList()
{
    this.head = null;
    this.tail = null;
}

// Printing the linked list
public void printList()
{
    Node ref = this.head;

    while(ref != null)
    {
        System.out.print(ref.getNum() + " ");
        Ref = ref.getNext();
    }
}

```

```

    }
    System.out.println();
}

// determine if the list is empty
public boolean isEmpty()
{
    if(head == null)
        return true;
    return false;
}

// Delete all the occurrences of number n from the list
public void delete(int n)
{
    if(!this.isEmpty())
    {
        while(this.getHead() == n)
        {
            this.deleteHead();
        }

        Node temp = head;
        while(temp.getNext() != null)
        {
            if(temp.getNext().getNum() == n)
            {
                Node t = temp.getNext();
                temp.setNext(t.getNext());

                if (t==tail)
                    tail = temp;
            }
            temp = temp.getNext();
        }
    }
}

```

There are a few methods of `LinkedList` that we have not implemented. Try to implement them as an exercise.