Linked List Implementation in Java

We know that the `LinkedList` class in Java is a doubly-linked list implementation of the `List` interface. This post provides an overview of common techniques to implement a linked list in Java programming language.

We know that each node of a linked list contains a single data field and a reference to the next node in the list. The nodes of the linked list are allocated in the heap memory during runtime by the JVM. We can use the constructor of the `Node` class to initialize the `data` field and the `next` pointer.

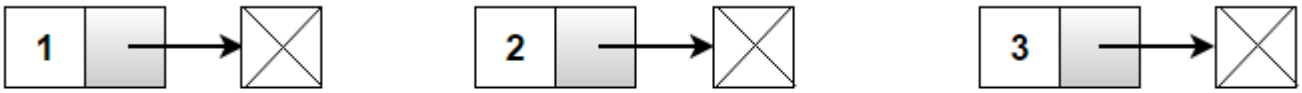
```
// A Linked List Node
class Node
{
    int data;
    Node next;

    // constructor
    Node(int data, Node next)
    {
        this.data = data;
        this.next = next;
    }
}
```

Practice This Problem

There are several methods to construct a singly linked list in Java:

1. Naive method

A simple solution would be to allocate memory for all individual nodes of the linked list, set their



Construct three linked list nodes

Rearrange the pointers to construct a list

```java
// A Linked List Node
class Node
{
    int data;
    Node next;

    Node(int data)
    {
        this.data = data;
        this.next = null;
    }
}

class Main
{
    // Helper function to print a given linked list
    public static void printList(Node head)
    {
        Node ptr = head;
        while (ptr != null)
        {
            System.out.print(ptr.data + " -> ");
            ptr = ptr.next;
        }

        System.out.println("null");
    }

    // Naive function for linked list implementation containing three nodes
    public static Node constructList()
    {
        // construct linked list nodes
        Node first = new Node(1);
        Node second = new Node(2);
        Node third = new Node(3);
        Node fourth = new Node(4);

        // rearrange the references to construct a list
        Node head = first;
        first.next = second;
        second.next = third;
        third.next = fourth;

        // return reference to the first node in the list
```

```
    {
        // `head` points to the head node of the linked list
        Node head = constructList();

        // print linked list
        printList(head);
    }
}
```

Download Run Code

We can write the above code in a single line by passing the next node as an argument to the `Node` constructor:

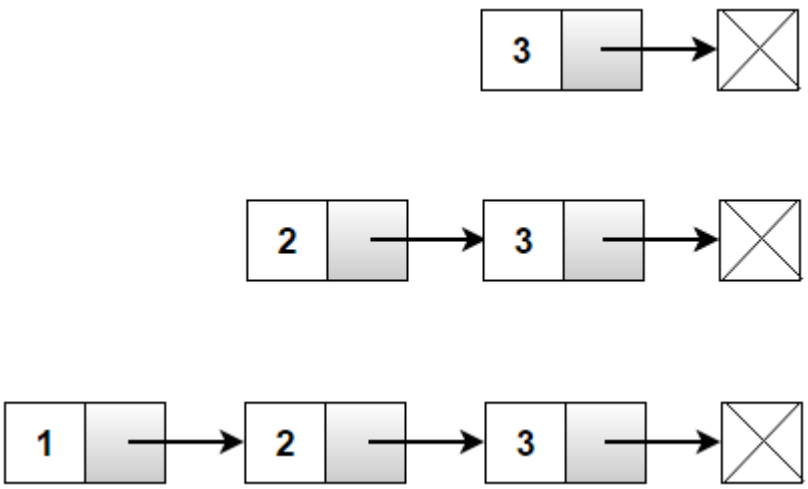
```
// A Linked List Node
class Node
{
    int data;
    Node next;

    Node(int data, Node next_node)
    {
        // Set data
        this.data = data;

        // set the next field to point to a given node of the list
        this.next = next_node;
    }
}

class Main
{
    // Helper function to print a given linked list
    public static void printList(Node head)
    {
        Node ptr = head;
        while (ptr != null)
        {
            System.out.print(ptr.data + " -> ");
            ptr = ptr.next;
        }

        System.out.println("null");
    }

    // Naive function for linked list implementation containing three nodes
    public static Node constructList()
    {
        Node head = new Node(1, new Node(2, new Node(3, null)));
        return head;
    }

    public static void main(String[] args)
    {
        // `head` points to the head node of the linked list
        Node head = constructList();
```

```
47  }
```

Download Run Code

2. Return Head Node

The standard solution adds a single node to the head end of any list. This function is called `push()` since we are adding the link to the head end, making a list look a bit like a stack.



This is demonstrated below, where we return the head node from the `push()` function and update the head in the caller.

```
1   // A Linked List Node
2   class Node
3   {
4       int data;
5       Node next;
6   }
7
8   class Main
9   {
10      // Helper function to print a given linked list
11      public static void printList(Node head)
12      {
13          Node ptr = head;
14          while (ptr != null)
15          {
16              System.out.print(ptr.data + " -> ");
17              ptr = ptr.next;
18          }
19          System.out.println("null");
20      }
21
22      public static Node push(Node head, int data)
```

```
27
28          // set the next field of the new node to point to the current
29          // first node of the list.
30
31          newNode.next = head;
32
33          // change the head to point to the new node, so it is
34          // now the first node in the list.
35
36          return newNode;
37      }
38
39      // Function for linked list implementation from the given set of keys
40      public static Node constructList(int[] keys)
41      {
42          Node head = null;
43
44          // start from the end of the array
45          for (int i = keys.length - 1; i >= 0; i--) {
46              head = push(head, keys[i]);
47          }
48
49          return head;
50      }
51
52      public static void main(String[] args)
53      {
54          // input keys
55          int[] keys = { 1, 2, 3, 4 };
56
57          // points to the head node of the linked list
58          Node head = constructList(keys);
59
60          // print linked list
61          printList(head);
62      }
63  }
```

Download Run Code

3. Make head reference global

We can construct a linked list by making the head reference global, but this approach is not recommended since **global variables** are usually considered bad practice.

```
1   // A Linked List Node
2   class Node
3   {
4       int data;
5       Node next;
6   }
7
8   class Main
9   {
10      // Helper function to print a given linked list
```

```
14          while (ptr != null)
15          {
16              System.out.print(ptr.data + " -> ");
17              ptr = ptr.next;
18          }
19          System.out.println("null");
20      }
21
22      // global head
23      public static Node head;
24
25      // Takes a list and a data value, creates a new link with the given
26      // data and pushes it onto the list's front.
27      public static Node push(int data)
28      {
29          // allocate a new node in a heap and set its data
30          Node newNode = new Node();
31          newNode.data = data;
32
33          // set the next field of the new node to point to the current
34          // head node of the list.
35          newNode.next = head;
36
37          // change the head to point to the new node, so it is
38          // now the first node in the list.
39
40          return newNode;
41      }
42
43      // Function for linked list implementation from the given set of keys
44      public static void constructList(int[] keys)
45      {
46          // start from the end of the array
47          for (int i = keys.length - 1; i >= 0; i--) {
48              head = push(keys[i]);
49          }
50      }
51
52      public static void main(String[] args)
53      {
54          // input keys
55          int[] keys = { 1, 2, 3, 4 };
56
57          // points to the head node of the linked list
58          constructList(keys);
59
60          // print linked list
61          printList(head);
62      }
63  }
```

Download Run Code

Continue Reading:

Linked List – Insertion at Tail | C, Java, and Python Implementation