DEBOZ DEN

UNLEASH YOUR MIND

HOME

« Back to Home (http://debozden.webs.com/apps/blog/) « Older Entry (http://debozden.webs.com/apps/blog/show/prev?from_id=4331332) | Newer Entry » (http://debozden.webs.com/apps/blog/show/next?from_id=4331332)

Double Linked List in Java (http://debozden.webs.com/apps/blog/show/4331332-double-linked-list-in-java)

(http://debozden.webs.com/apps/profile/55861244/) Posted by Debabrata Podder (http://debozden.webs.com/apps/profile/55861244/) on July 22, 2010 at 1:05 PM

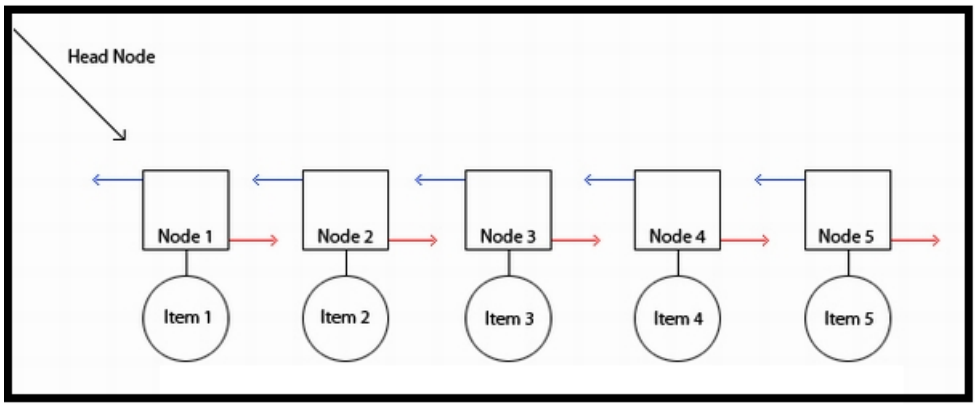
Perhaps data structure is the most important aspect of any programming language. Manipulation of data structure in procedural language like C is quite different than java. Especially java does not allow pointer, this restricts some features: mostly performance. Pointer is the fastest way to access memory where as java has to create Object and uses pass by value to work with it. Let's see what data structure is:

From the definition we get: **data structure** is a way of storing and organizing data in a system for efficient usage.

As far as the term "efficient" arises, let's see the efficiency of java and C. well, C has tremendous power of a feature called, "pointer" which access memory by reference, more precisely. Pointer is a variable which points out the memory address of any stored data! The quickest and devastating indeed! Using pointer you can access some other data's memory which is outside of your program scope. Whereas java thinks this is absolute violation of security, so it eliminated pointer aspect and uses pass by value. Common misconception is java allows "pass by reference" as we create reference of object and can pass it, but the truth is value of variable in java that is passed is either primitive or reference but not Object itself. So when we pass reference, we actually do not change the value of Object, rather copy of the object's reference.

Anyway, creating object is costly, and slow process than handling actual data, in this case, java is slower than C.

Here is the feature of highly regarded data structure: double linked list which has one header (that points to previous data) one body (storing data) and one tail (that points to next data). These 3 things comprise a complete node and several such nodes make a double linked list.



The list starts with a header and ends with tail node as well. Here is a program to manipulate double linked list in Java

```
package doublelinkedlist;

/**

 * @author TITTU

 */

public class Demo {

   private Links first;

   private Links last;

   private static int count = 0;

   public Demo() {

      first = null;
```

```
        last = null;
    }
    public boolean isEmpty() {
        return first == null;
    }
    public void insertFirst(Object dd) {
        Links newLink = new Links(dd);
        if (isEmpty()) {
            last = newLink;
        } else {
            first.previous = newLink;
        }
        newLink.next = first;
        first = newLink;
        count++;
    }
    public void insertLast(Object dd) {
        Links newLink = new Links(dd);
        if (isEmpty()) {
            first = newLink;
        } else {
            last.next = newLink;
            newLink.previous = last;
        }
        last = newLink;
        count++;
    }
    public void elementAt(int index) {
        Links current = first; // start at beginning
        if (current == null) {
            System.out.println("No Element found");
        } else {
            for (int i = 0; i < index; i++){ // until end of list,
                current = current.next; // move to next link
            }
            current.displayLink();
            System.out.println("");
        }
    }
    public int getIndex(Object key) {
```

```
    int index = 1;
    Links current = first; // start at beginning
    while (current.dData != key){ // until end of list,
        current = current.next; // move to next link
        index++;
    }
    return index;
}
public void insertAt(int index, Object data) {
    Links current = first; // start at beginning
    if (current == null) {
        System.out.println("No Element found");
    } else {
        for (int i = 0; i < index; i++){ // until end of list,
            current = current.next; // move to next link
        }
    }
    insertAfter(current, data);
}
public void deleteAt(int index) {
    Links current = first; // start at beginning
    if (current == null) {
        System.out.println("No Element found");
    } else {
        for (int i = 0; i < index; i++){ // until end of list,
            current = current.next; // move to next link
        }
    }
    deleteKey(current);
}
public Links deleteFirst() {
    Links temp = first;
    if (first.next == null) {
        last = null;
    } else {
        first.next.previous = null;
    }
    first = first.next;
    count--;
    System.out.println("First Item Deleted");
```

```
        return temp;
    }
    public Links deleteLast() {
        Links temp = last;
        if (first.next == null) {
            first = null;
        } else {
            last.previous.next = null;
        }
        last = last.previous;
        count--;
        System.out.println("Last item Deleted");
        return temp;
    }
    public boolean insertAfter(Object key, Object dd) {
        Links current = first;
        while (current.dData != key) {
            current = current.next;
            if (current == null) {
                return false; // cannot find it
            }
        }
        Links newLink = new Links(dd); // make new link
        if (current == last){ // if last link,
            newLink.next = null;
            last = newLink;
        } else{ // not last link,
            newLink.next = current.next;
            current.next.previous = newLink;
        }
        newLink.previous = current;
        current.next = newLink;
        count++;
        return true; // found it, insert
    }
    public Links deleteKey(Object key) {
        Links current = first;
        while (current.dData != key) {
            current = current.next;
            if (current == null) {
```

```
            return null; // cannot find it
        }
    }
    if (current == first) {// found it; first item?
        first = current.next;
    } else {
        current.previous.next = current.next;
    }
    if (current == last) {// last item?
        last = current.previous;
    } else {// not last
        current.next.previous = current.previous;
    }
    count--;
    return current; // return value
}
public void displayForward() {
    System.out.print("List (first to last): ");
    Links current = first; // start at beginning
    while (current != null) { // until end of list,
        current.displayLink();
        current = current.next; // move to next link
    }
    System.out.println("");
}
public void displayBackward() {
    System.out.print("Reversed List : ");
    Links current = last;
    while (current != null) {
        current.displayLink();
        current = current.previous;
    }
    System.out.println("");
}
public static void main(String[] args) {
    Demo theList = new Demo();
    theList.insertFirst(22);
    theList.insertFirst(44);
    theList.insertLast(33);
    theList.insertLast(55);
```

```
        System.out.println("Element at : position 3 ");
        theList.elementAt(3);
        System.out.println("index of 44 = " + theList.getIndex(22));
        theList.insertAt(2, 77);
        theList.displayForward();
        theList.displayBackward();
        theList.deleteFirst();
        theList.deleteLast();
        theList.deleteKey(11);
        theList.displayForward();
        theList.insertAfter(22, 77); // insert 77 after 22
        theList.insertAfter(33, 88 ); // insert 88 after 33
        theList.displayForward();
    }
}
class Links {
    public Object dData; // data item
    public Links next; // next link in list
    public Links previous; // previous link in list
    public Links(Object d) {
        dData = d;
    }
    public void displayLink() {
        System.out.print(dData.toString() + " ");
    }
}
```

Categories: Salt in Dish (http://debozden.webs.com/apps/blog/categories/show/583555-salt-in-dish)

Post a Comment

OOPS!

Oops, you forgot something.

OOPS!

The words you entered did not match the given text. Please try again.

Name

Already a member? Sign In (http://debozden.webs.com/apps/auth/login)

Email

Message

1 Comment

(http://debozden.webs.com/apps/profile/55861244/)

Debabrata Podder (http://debozden.webs.com/apps/profile/55861244/)

07:25 AM on June 07, 2011

Well you can directly use LinkedList Collection for double link list implementation. Java Collection framework allows linked lists to be used as a stack, queue, or deque. This tutorial is for manual implementation of linked list

BOOKMARK (http://www.addthis.com/bookmark.php?v=120&winname=addthis&pub=webs&source=men-120&lng=en&s=&url=http%3A%2F%2Fdebozden.webs.com%2Fapps%2Fblog%2Fshow%2F4331332-double-linked-list-in-java&title=Double%20Linked%20List%20in%20Java%20-%20Deboz%20Den&logo=&logobg=&logocolor=&ate=AT-webs/-/-/522d10e9e7670b12/2/52032cb79e304927&frommenu=1&uid=52032cb79e304927&ufbl=1&ct=0&pre=http%3A%2F%2Fdebozden.webs.com%2Fnotify-DETE_Student_Uncategorised_Allow_Page%3FaHR0cDovL2RlYm96ZGVuLndlYnMuY29tL2FwcHMvYmxvZy9zaG93LzQzMzEzMzItZG91YmxlLWxpbmtlZC1saXN0LWluLWphdmE%3D&tt=0&captcha_provider=nucaptcha)

Members Area

Sign In or Register

Categories

Salt in Dish (9)

me, myself and ... (1)

Recent Blog Entries

Design Pattern - Observer (Trigger in Java!!!) 1
by Debabrata Podder | 0 comments

Design Pattern - Observer (Trigger in Java!!!) 2
by Debabrata Podder | 0 comments

Design Pattern - Observer (Trigger in Java!!!) 3
by Debabrata Podder | 0 comments

Design Pattern - Observer (Trigger in Java!!!) 4
by Debabrata Podder | 0 comments

Recent Photos






Recent Videos

Space
201 views - 0 comments

Quantum Leap

233 views - 0 comments

Multiverse

191 views - 0 comments

Time

212 views - 0 comments
