DoublyLinkedList.java

Below is the syntax highlighted version of DoublyLinkedList.java from §1.3 Stacks and Queues.

```
/*************************************************************************
 *  Compilation:  javac DoublyLinkedList.java
 *  Execution:    java DoublyLinkedList
 *
 *  A list implemented with a doubly linked list. The elements are stored
 *  (and iterated over) in the same order that they are inserted.
 *
 *  % java DoublyLinkedList 10
 *  10 random integers between 0 and 99
 *  24 65 2 39 86 24 50 47 13 4
 *
 *  add 1 to each element via next() and set()
 *  25 66 3 40 87 25 51 48 14 5
 *
 *  multiply each element by 3 via previous() and set()
 *  75 198 9 120 261 75 153 144 42 15
 *
 *  remove elements that are a multiple of 4 via next() and remove()
 *  75 198 9 261 75 153 42 15
 *
 *  remove elements that are even via previous() and remove()
 *  75 9 261 75 153 15
 *
 *************************************************************************/

import java.util.ListIterator;
import java.util.NoSuchElementException;

public class DoublyLinkedList<Item> implements Iterable<Item> {
    private int N;        // number of elements on list
    private Node pre;     // sentinel before first item
    private Node post;    // sentinel after last item

    public DoublyLinkedList() {
        pre  = new Node();
        post = new Node();
        pre.next = post;
        post.prev = pre;
    }

    // linked list node helper data type
    private class Node {
        private Item item;
        private Node next;
        private Node prev;
    }

    public boolean isEmpty()    { return N == 0; }
    public int size()           { return N;      }

    // add the item to the list
    public void add(Item item) {
        Node last = post.prev;
        Node x = new Node();
        x.item = item;
        x.next = post;
        x.prev = last;
        post.prev = x;
        last.next = x;
        N++;
    }

    public ListIterator<Item> iterator()  { return new DoublyLinkedListIterator(); }

    // assumes no calls to DoublyLinkedList.add() during iteration
    private class DoublyLinkedListIterator implements ListIterator<Item> {
        private Node current      = pre.next;  // the node that is returned by next()
        private Node lastAccessed = null;      // the last node to be returned by prev() or
                                               // reset to null upon intervening remove() o
        private int index = 0;

        public boolean hasNext()      { return index < N; }
        public boolean hasPrevious()  { return index > 0; }
        public int previousIndex()    { return index - 1; }
        public int nextIndex()        { return index;     }

        public Item next() {
            if (!hasNext()) throw new NoSuchElementException();
            lastAccessed = current;
            Item item = current.item;
            current = current.next;
            index++;
            return item;
        }

        public Item previous() {
            if (!hasPrevious()) throw new NoSuchElementException();
            current = current.prev;
            index--;
            lastAccessed = current;
            return current.item;
        }

        // replace the item of the element that was last accessed by next() or previous()
        // condition: no calls to remove() or add() after last call to next() or previous()
        public void set(Item item) {
            if (lastAccessed == null) throw new IllegalStateException();
            lastAccessed.item = item;
        }

        // remove the element that was last accessed by next() or previous()
        // condition: no calls to remove() or add() after last call to next() or previous()
        public void remove() {
            if (lastAccessed == null) throw new IllegalStateException();
            Node x = lastAccessed.prev;
            Node y = lastAccessed.next;
            x.next = y;
            y.prev = x;
            N--;
            if (current == lastAccessed)
                current = y;
            else
                index--;
            lastAccessed = null;
        }

        // add element to list
        public void add(Item item) {
            Node x = current.prev;
            Node y = new Node();
            Node z = current;
            y.item = item;
            x.next = y;
            y.next = z;
            z.prev = y;
            y.prev = x;
            N++;
            index++;
            lastAccessed = null;
        }

    }

    public String toString() {
        StringBuilder s = new StringBuilder();
        for (Item item : this)
            s.append(item + " ");
        return s.toString();
    }

    // a test client
    public static void main(String[] args) {
        int N  = Integer.parseInt(args[0]);

        // add elements 1, ..., N
        StdOut.println(N + " random integers between 0 and 99");
        DoublyLinkedList<Integer> list = new DoublyLinkedList<Integer>();
        for (int i = 0; i < N; i++)
            list.add((int) (100 * Math.random()));
        StdOut.println(list);
        StdOut.println();

        ListIterator<Integer> iterator = list.iterator();

        // go forwards with next() and set()
        StdOut.println("add 1 to each element via next() and set()");
        while (iterator.hasNext()) {
            int x = iterator.next();
            iterator.set(x + 1);
        }
        StdOut.println(list);
        StdOut.println();

        // go backwards with previous() and set()
        StdOut.println("multiply each element by 3 via previous() and set()");
        while (iterator.hasPrevious()) {
            int x = iterator.previous();
            iterator.set(x + x + x);
        }
        StdOut.println(list);
        StdOut.println();


        // remove all elements that are multiples of 4 via next() and remove()
        StdOut.println("remove elements that are a multiple of 4 via next() and remove()");
        while (iterator.hasNext()) {
            int x = iterator.next();
            if (x % 4 == 0) iterator.remove();
        }
        StdOut.println(list);
        StdOut.println();


        // remove all even elements via previous() and remove()
        StdOut.println("remove elements that are even via previous() and remove()");
        while (iterator.hasPrevious()) {
            int x = iterator.previous();
            if (x % 2 == 0) iterator.remove();
        }
        StdOut.println(list);
        StdOut.println();


        // add elements via next() and add()
        StdOut.println("add elements via next() and add()");
        while (iterator.hasNext()) {
            int x = iterator.next();
            iterator.add(x + 1);
        }
        StdOut.println(list);
        StdOut.println();

        // add elements via previous() and add()
        StdOut.println("add elements via previous() and add()");
        while (iterator.hasPrevious()) {
            int x = iterator.previous();
            iterator.add(x * 10);
            iterator.previous();
        }
        StdOut.println(list);
        StdOut.println();
    }
}
```