import static java.lang.System.*;

/**
 * Class LinkedList defines an singly-linked list of integer-valued nodes.
 */

public class LinkedList {

    /** Pointer to the head of the list */
    ListNode head;

    /** Current number of elements in the list */
    int length;

    /**
     * Allocate a new empty list, with null head pointer and length = 0.
     */
    LinkedList() {
        /* Initialize the head and length. */
        head = null;
        length = 0;
    }

    /**
     * Insert the given node before the given index position i in the given
     * list, for 0 ≤ i ≤ list.length. Do nothing if i < 0 or i >
     * list.length. If node was inserted, increment list.length by 1.
     */
    public void insert(ListNode node, int i) {
        ListNode splice_node; /* pointer to splice-in position */
        /* Do nothing if i is out of range. */
        if (i < 0 || i > length) {
            return;
        }
        /*
         * Node will go somewhere, so increment length.
         */
        length++;
        /* If the list is empty, put the element at the head.
         */
        if (length == 0) {
            head = node;
        }
        /*
         * If i = 0, splice the node in at the head.
         */
        else if (i == 0) {
            node.next = head;
            head = node;
        }
        /*
         * Otherwise, splice the node in before the given position.
         */
        else {
            splice_node = getIthNode(i-1);
            node.next = splice_node.next;
            splice_node.next = node;
        }
    }

    /**
     * Return the ith node in this. Return null if the list is empty or i < 0
     * or i >= this.length.
     */
    ListNode getIthNode(int i) {
        ListNode node = null; /* Return value */
        int j; /* Search index */
        /*
         * Outta here if list is empty, i<0, or i>=length.
         */
        if (length == 0 || i < 0 || i >= length) {
            return null;
        }
        /*
         * Traverse the list with a for loop. Note that there's nothing to do
         * in the loop body, since the bounds checks have already been taken
         * care of.
         */
        for (node = head, j = 0; j < i; node = node.next, j++) ;
        /*
         * Return the located node.
         */
        return node;
    }

    /**
     * Print to stdout the elements of the given list, comma separated, in list
     * order, with a newline at the end.
     */
    public void printList() {
        /*
         * Traverse the list with a for loop. Note that there's nothing to do
         * in the loop body, since the bounds checks have already been taken
         * care of.
         */
        for (node = head, j = 0; j < i; node = node.next, j++) ;
        /*
         * Return the located node.
         */
    }
}
public void printList() {

    /* traversal pointer */
    ListNode node;

    /* Traverse the list, printing a comma after all but the last element. */
    for (node = head; node != null; node = node.next) {
        out.printf("%d%s", node.value, node.next != null ? "," : ");
    }
    out.println();
}