```java
public class Singlylist {
    Node head;

    static class Node {
        int value;
        Node next;

        Node(int val) {
            this.value = val;
            // this.next = null;
        }
    }

    void printList() {
        StringBuilder ret_str = new StringBuilder("[");
        Node last = this.head;
        while (last != null) {
            ret_str.append(last.value).append(", ");
            last = last.next;
        }
        if (ret_str.length() > 1) {
            ret_str.setLength(ret_str.length() - 2); // Remove trailing comma and space
        }
        ret_str.append("]");
        System.out.println(ret_str);
    }

    void push(int val) {
        Node new_node = new Node(val);

        // If the list is empty, set the new node as the head
        if (this.head == null) {
            this.head = new_node;
            return;
        }

        // Otherwise, traverse to the end of the list and add the new node
        Node last = this.head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = new_node;
    }

    void insert(int val, int idx) {
        Node new_node = new Node(val);

        // Insert at the head if index is 0
        if (idx == 0) {
            new_node.next = this.head;
            this.head = new_node;
            return;
        }

        // Traverse to the specified index
        Node last = this.head;
        Node prev = null;
        int counter = 0;
        while (last != null && counter < idx) {
            prev = last;
            last = last.next;
            counter++;
        }

        // Insert the new node at the specified index
        if (prev != null) {
            new_node.next = last;
            prev.next = new_node;
        } else {
            throw new IndexOutOfBoundsException("Invalid Index! No Insert");
        }
    }

    void remove(int idx) {
        // Remove the head node if index is 0
        if (idx == 0) {
            if (this.head != null) {
                this.head = this.head.next;
            } else {
                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
            }
            return;
        }

        // Traverse to the specified index
        Node last = this.head;
        Node prev = null;
        int counter = 0;
        while (last != null && counter < idx) {
            prev = last;
            last = last.next;
            counter++;
        }

        // Remove the node at the specified index
        if (last != null) {
            prev.next = last.next;
        } else {
            throw new IndexOutOfBoundsException("Invalid Index! No Delete");
        }
    }

    void pop() {
        if (this.head == null) {
            throw new IndexOutOfBoundsException("Invalid Index! No Delete");
        }
        if (this.head.next == null) {
            this.head = null;
            return;
        }
        Node last = this.head;
        Node prev = null;
        while (last.next != null) {
            prev = last;
            last = last.next;
        }
        System.out.println("Popped: " + last.value);

        prev.next = null;
    }

    public static void main(String[] args) {
        Singlylist li = new Singlylist();
        li.push(10);
        li.push(20);
        // li.push(30);
        // li.push(40);

        // Uncomment to test insertions
        // li.insert(50, 0);
        // li.insert(100, 3);
        // li.insert(3434, 232); // This will throw an exception

        // li.remove(0);
        // li.remove(213); // This will throw an exception

        // li.pop();
        li.pop();
        // li.pop();
        li.printList(); // Output: [20, 30, 40]
    }

}
// * doubly
class Doublylist {
    Node head;

    static class Node {
        int value;
        Node next;
```

```java
        Node prev; // # changes in double

        Node(int val) {
            this.value = val;
            this.next = null;
            this.prev = null; // # changes in double
        }
    }

    void printList() {
        StringBuilder ret_str = new StringBuilder("[");
        Node last = this.head;
        while (last != null) {
            ret_str.append(last.value).append(", ");
            last = last.next;
        }
        if (ret_str.length() > 1) {
            ret_str.setLength(ret_str.length() - 2); // Remove trailing comma and space
        }
        ret_str.append("]");
        System.out.println(ret_str);
    }

    void push(int val) {
        Node new_node = new Node(val);

        // If the list is empty, set the new node as the head
        if (this.head == null) {
            this.head = new_node;
            return;
        }

        // Otherwise, traverse to the end of the list and add the new node
        Node last = this.head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = new_node;
        new_node.prev = last; // # changes in double
    }

    void insert(int val, int idx) {
        Node new_node = new Node(val);

        // Insert at the head if index is 0
        if (idx == 0) {
            new_node.next = this.head;
            if (this.head != null) {
                this.head.prev = new_node; // # changes in double
            }
            this.head = new_node;
            return;
        }

        // Traverse to the specified index
        Node last = this.head;
        Node prev = null;
        int counter = 0;
        while (last != null && counter < idx) {
            prev = last;
            last = last.next;
            counter++;
        }

        // Insert the new node at the specified index
        if (prev != null) {
            new_node.next = last;
            new_node.prev = prev; // # changes in double
            prev.next = new_node;
            if (last != null) {
                last.prev = new_node; // # changes in double
            }
        } else {
            throw new IndexOutOfBoundsException("Invalid Index! No Insert");
        }
    }

    void remove(int idx) {
        // Remove the head node if index is 0
        if (idx == 0) {
            if (this.head != null) {
                this.head = this.head.next;
                if (this.head != null) {
                    this.head.prev = null; // # changes in double
                }
            } else {
                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
            }
            return;
        }

        // Traverse to the specified index
        Node last = this.head;
        Node prev = null;
        int counter = 0;
        while (last != null && counter < idx) {
            prev = last;
            last = last.next;
            counter++;
        }
```

```java

        // Remove the node at the specified index
        if (last != null) {
            prev.next = last.next;
            if (last.next != null) {
                last.next.prev = prev; // # changes in double
            }
        } else {
            throw new IndexOutOfBoundsException("Invalid Index! No Delete");
        }
    }

    void pop() {
        if (this.head == null) {
            throw new IndexOutOfBoundsException("Invalid Index! No Delete");
        }
        if (this.head.next == null) {
            this.head = null;
            return;
        }
        Node last = this.head;
        while (last.next != null) {
            last = last.next;
        }
        System.out.println("Popped: " + last.value);

        if (last.prev != null) {
            last.prev.next = null; // # changes in double
        }
    }

    public static void main(String[] args) {
        Doublylist list = new Doublylist();
        list.push(1);
        list.push(2);
        list.push(3);
        list.printList(); // Output: [1, 2, 3]
        list.insert(4, 1);
        list.printList(); // Output: [1, 4, 2, 3]
        list.remove(2);
        list.printList(); // Output: [1, 4, 3]
        list.pop();
        list.printList(); // Output: [1, 4]
    }
}

// * circular
public class Circularlist {
    Node head;

    static class Node {
        int value;
        Node next;

        Node(int val) {
            this.value = val;
            // this.next = null;
        }
    }

    void printList() {
        StringBuilder ret_str = new StringBuilder("[");
        if (this.head != null) { // # changes in circular
            Node last = this.head;
            do {
                ret_str.append(last.value).append(", ");
                last = last.next;
            } while (last != this.head); // # changes in circular
            if (ret_str.length() > 1) {
                ret_str.setLength(ret_str.length() - 2); // Remove trailing comma and space
            }
        }
        ret_str.append("]");
        System.out.println(ret_str);
    }

    void push(int val) {
        Node new_node = new Node(val);

        // If the list is empty, set the new node as the head
        if (this.head == null) {
            this.head = new_node;
            new_node.next = this.head; // # changes in circular
            return;
        }

        // Otherwise, traverse to the end of the list and add the new node
        Node last = this.head;
        while (last.next != this.head) { // # changes in circular
            last = last.next;
        }
        last.next = new_node;
        new_node.next = this.head; // # changes in circular
    }

    void insert(int val, int idx) {
        Node new_node = new Node(val);

        // Insert at the head if index is 0
        if (idx == 0) {
            if (this.head == null) {
                this.head = new_node;
                new_node.next = this.head; // # changes in circular
            } else {
                Node last = this.head;
                while (last.next != this.head) { // # changes in circular
                    last = last.next;
                }
                new_node.next = this.head;
                this.head = new_node;
                last.next = this.head; // # changes in circular
            }
            return;
        }

        // Traverse to the specified index
        Node last = this.head;
        Node prev = null;
        int counter = 0;
        do {
            prev = last;
            last = last.next;
            counter++;
        } while (last != this.head && counter < idx); // # changes in circular

        // Insert the new node at the specified index
        if (prev != null && counter == idx) {
            new_node.next = last;
            prev.next = new_node;
        } else {
            throw new IndexOutOfBoundsException("Invalid Index! No Insert");
        }
    }

    void remove(int idx) {
        // Remove the head node if index is 0
        if (idx == 0) {
            if (this.head != null) {
                Node last = this.head;
                while (last.next != this.head) { // # changes in circular
                    last = last.next;
                }
                if (this.head.next == this.head) { // Only one node in the list
                    this.head = null;
                } else {
                    this.head = this.head.next;
                    last.next = this.head; // # changes in circular
                }
            } else {
                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
            }
```

```java
            return;
        }

        // Traverse to the specified index
        Node last = this.head;
        Node prev = null;
        int counter = 0;
        do {
            prev = last;
            last = last.next;
            counter++;
        } while (last != this.head && counter < idx); // # changes in circular

        // Remove the node at the specified index
        if (last != this.head && counter == idx) {
            prev.next = last.next;
        } else {
            throw new IndexOutOfBoundsException("Invalid Index! No Delete");
        }
    }

    void pop() {
        if (this.head == null) {
            throw new IndexOutOfBoundsException("Invalid Index! No Delete");
        }
        if (this.head.next == this.head) { // Only one node in the list
            this.head = null;
            return;
        }
        Node last = this.head;
        Node prev = null;
        while (last.next != this.head) { // # changes in circular
            prev = last;
            last = last.next;
        }
        System.out.println("Popped: " + last.value);

        if (prev != null) {
            prev.next = this.head; // # changes in circular
        }
    }

    public static void main(String[] args) {
        Circularlist li = new Circularlist();
        li.push(10);
        li.push(20);
        li.push(30);
        li.push(40);

        // Uncomment to test insertions
        li.insert(50, 0);
        li.insert(100, 3);
        // li.insert(3434, 232); // This will throw an exception

        li.remove(0);
        // li.remove(213); // This will throw an exception

        li.pop();
        li.printList(); // Output: [10, 20, 30]
    }
}
```