Solution 3 – Iterator Design Pattern

```
package iterator;

import java.util.*;

public class MyList extends AbstractCollection {

  private class Element {
     Object data;
     Element next;
     Element(Object d, Element n) {data=d; next=n;}
  }

  protected Element head = null;
  protected Element tail = null;
  protected int count = 0;

  public boolean add(Object o) {
    if (tail==null) {
       head = tail = new Element(o,null);
    } else {
       tail = tail.next = new Element(o,null);
    }
    ++count;
    return true;
  }

  public boolean remove(Object o) {
    if (head==null) return false;
    if (o.equals(head.data)) {
        head=head.next;
        --count;
        if (count==0) tail=null;
        return true;
    }
    Element e1= head.next;
    Element e2 = head;
    while(e1!=null) {
       if (o.equals(e1.data)){
          e2.next = e1.next;
          if (e1==tail) tail = e2;
          --count;
          return true;
        }
        e2=e1;
        e1=e1.next;
    }
    return false;
  }

  public Iterator iterator() {
    return new Itr();
  }

  public int size() {return count;}

  private class Itr implements Iterator {
    private Element current;

    Itr() {
      current = head;
    }

    public boolean hasNext() {
      return current!=null;
    }

    public Object next() {
       if (current == null)
              throw new NoSuchElementException();
       Object o = current.data;
       current = current.next;
       return o;
    }

    public void remove() {
      throw new UnsupportedOperationException();
    }
  }

}
```