

Double-Linked List

I. Java Implementation

getFirst

```
@Assertion(value = { @Case(
pre = "repOK() && size >0",
post = "repOK() ") })
public Object getFirst() {
    if (size==0)
        throw new NoSuchElementException();
    return header.next.element;
}
```

removeLast

```
@Assertion(value = { @Case(
pre = "repOK()",
post = "repOK() ") })
public Object removeLast() {
    Object last = header.previous.element;
    remove(header.previous);
    return last;
}
```

remove

```
@Assertion(value = { @Case(
pre = "repOK()",
post = "repOK()") })
public boolean remove(Object o) {
    if (o==null) {
        for (Entry e = header.next; e != header; e = e.next) {
            if (e.element==null) {
                remove(e);
                return true;
            }
        }
    } else {
        for (Entry e = header.next; e != header; e = e.next) {
            if (o==e.element) {
                remove(e);
                return true;
            }
        }
    }
    return false;
}
```

```

        }
    }
}
return false;
}

```

clear

```

@Assertion(value = { @Case(
pre = "repOK()",
post = "repOK() && size==0 ") })
public void clear() {
    modCount++;
    header.next = header.previous = header;
    size = 0;
}

```

indexOf

```

@Assertion(value = { @Case(
pre = "repOK()",
post = "repOK()") })
public int indexOf(Object o) {
    int index = 0;
    if (o==null) {
        for (Entry e = header.next; e != header; e = e.next) {
            if (e.element==null)
                return index;
            index++;
        }
    } else {
        for (Entry e = header.next; e != header; e = e.next) {
            if (o==e.element)
                return index;
            index++;
        }
    }
    return -1;
}

```

lastIndexOf

```

@Assertion(value = { @Case(
pre = "repOK()",
post = "repOK()") })
public int lastIndexOf(Object o) {
    int index = size;
    if (o==null) {
        for (Entry e = header.previous; e != header; e = e.previous) {
            index--;
            if (e.element==null)
                return index;
        }
    } else {
        for (Entry e = header.previous; e != header; e = e.previous) {
            index--;
            if (o==e.element)
                return index;
        }
    }
    return -1;
}

```

addBefore

```

@Assertion(value = { @Case(
pre = "repOK() && inList(e)",
post = "repOK()") })
private Entry addBefore(Object o, Entry e) {
    Entry newEntry = new Entry(o, e, e.previous);
    newEntry.previous.next = newEntry;
    newEntry.next.previous = newEntry;
    size++;
    modCount++;
    return newEntry;
}

```

toArray

```

@Assertion(value = { @Case(
pre = "repOK()",
post = "$ret !=null && $ret.length==size") })
public Object[] toArray() {
    Object[] result = new Object[size];
}

```

```
int i = 0;
for (Entry e = header.next; e != header; e = e.next)
    result[i++] = e.element;
return result;
}
```

repOK

```
boolean repOK() {
    if(header==null) {
        return false;
    }
    Entry tmp = header;
    int i=0;
    do {
        if(!tmp.nonNullPointers() || !tmp.repOK()) {
            return false;
        }
        tmp = tmp.next;
        if(tmp != header) {
            i++;
        }
    } while(tmp != header);
    tmp = header;
    return i==size;
}
```