Building Java Programs

Chapter 16
Linked Nodes

reading: 16.1
Recall: stacks and queues

- **stack**: retrieves elements in reverse order as added
- **queue**: retrieves elements in same order as added
Collection efficiency

• Complexity class of various operations on collections:

<table>
<thead>
<tr>
<th>Method</th>
<th>ArrayList</th>
<th>Stack</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>add (or push)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>add(index, value)</td>
<td>O(N)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>indexOf</td>
<td>O(N)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>get</td>
<td>O(1)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>remove</td>
<td>O(N)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>set</td>
<td>O(1)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>size</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

• Could we build lists differently to optimize other operations?
Array vs. linked structure

- All collections in this course use one of the following:
  - an **array** of all elements
    - examples: ArrayList, Stack, HashSet, HashMap
  
  | 42 | -3 | 17 | 9 |

  - **linked objects** storing a value and references to other(s)
    - examples: LinkedList, TreeSet, TreeMap

  front → 42 → -3 → 17 → 9

- First, we will learn how to create a **linked list**.
- To understand linked lists, we must understand references.
Memory for a List

- Array (contiguous in memory)
  
  | 42 | -3 | 17 | 9 |

- Spread in memory
  
  | 42 | 9 | -3 | 17 |
A list node class

```java
public class ListNode {
    int data;
    ListNode next;
}
```

- Each list node object stores:
  - one piece of integer data
  - a reference to another list node

- ListNode objects can be "linked" into chains to store a list of values:
References to same type

- What would happen if we had a class that declared one of its own type as a field?

```java
public class Strange {
    private String name;
    private Strange other;
}
```

- Will this compile?
  - If so, what is the behavior of the `other` field? What can it do?
  - If not, why not? What is the error and the reasoning behind it?
public class ConstructList1 {
    public static void main(String[] args) {
        ListNode list = new ListNode();
        list.data = 42;
        list.next = new ListNode();
        list.next.data = -3;
        list.next.next = new ListNode();
        list.next.next.data = 17;
        list.next.next.next = null;
        System.out.println(list.data + " " + list.next.data + " " + list.next.next.data);
        // 42 -3 17
    }
}
List node w/ constructor

public class ListNode {
    int data;
    ListNode next;

    public ListNode(int data) {
        this.data = data;
        this.next = null;
    }

    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}

- Exercise: Modify the previous client to use these constructors.
Linked node problem 1

- What set of statements turns this picture:

```
list  
10  
```

- Into this?

```
list  
10
20
30
```
References vs. objects

\[ \text{variable} = \text{value}; \]

A **variable** (left side of `=`) is an arrow (the base of an arrow)
A **value** (right side of `=`) is an object (a box; what an arrow points at)

- For the list at right:
  - `a.next = value;` means to adjust where 1 points
  - `variable = a.next;` means to make `variable` point at 2
Reassigning references

- when you say:
  - `a.next = b.next;`

- you are saying:
  - "Make variable `a.next` refer to the same value as `b.next`."
  - Or, "Make `a.next` point to the same place that `b.next` points."
Linked node problem 2

- What set of statements turns this picture:

  list
  ↓
  |  |
  |  v
  |    |
  | data | next |
  | 10   |      |
  |      |    → |
  |      |  |
  | data | next |
  | 20   |      |

- Into this?

  list
  ↓
  |  |
  |  v
  |    |
  | data | next |
  | 30   |      |
  |      |    → |
  |      |  |
  | data | next |
  | 10   |      |
  |      |    → |
  | data | next |
  | 20   |      |
Linked node problem 3

• What set of statements turns this picture:

<table>
<thead>
<tr>
<th>list1</th>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>list2</th>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>

• Into this?

<table>
<thead>
<tr>
<th>list1</th>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>list2</th>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
Linked node problem 3

- How many ListNode variables?

- Which variables change?
Linked node problem 3

- How many ListNode variables?

- Which variables change?

```python
list1.next.next = list2
```
Linked node problem 3

- How many ListNode variables?

- Which variables change?

```python
list1.next.next = list2
list2 = list2.next
```
Linked node problem 3

- How many ListNode variables?

- Which variables change?

```java
list1.next.next = list2
list2 = list2.next
list1.next.next.next = null
```
References vs. objects

variable = value;

a variable (left side of =) is an arrow (the base of an arrow)
a value (right side of =) is an object (a box; what an arrow points at)

• For the list at right:
  • a.next = value;
    means to adjust where 1 points
  • variable = a.next;
    means to make variable point at 2
list1

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

current
Linked node question

- Suppose we have a long chain of list nodes:

  - We don't know exactly how long the chain is.
  - How would we print the data values in all the nodes?
Algorithm pseudocode

- Start at the **front** of the list.
- While (there are more nodes to print):
  - Print the current node's **data**.
  - Go to the **next** node.

- How do we walk through the nodes of the list?

```python
list = list.next;  // is this a good idea?
```
Traversing a list?

- One (bad) way to print every value in the list:

```java
while (list != null) {
    System.out.println(list.data);
    list = list.next;  // move to next node
}
```

- What's wrong with this approach?
  - (It loses the linked list as it prints it!)
A current reference

- Don't change list. Make another variable, and change it.
  - A ListNode variable is NOT a ListNode object

```java
ListNode current = list;
```

- What happens to the picture above when we write:
  ```java
  current = current.next;
  ```
Traversing a list correctly

- The correct way to print every value in the list:

```java
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;  // move to next node
}
```

- Changing `current` does not damage the list.
Linked List vs. Array

• Print list values:

```java
ListNode list = ...;

ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```

• Similar to array code:

```java
int[] a = ...;

int i = 0;
while (i < a.length) {
    System.out.println(a[i]);
    i++;
}
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Array Code</th>
<th>Linked List Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to front of list</td>
<td>int i = 0;</td>
<td>ListNode current = list;</td>
</tr>
<tr>
<td>Test for more elements</td>
<td>i &lt; size</td>
<td>current != null</td>
</tr>
<tr>
<td>Current value</td>
<td>elementData[i]</td>
<td>current.data</td>
</tr>
<tr>
<td>Go to next element</td>
<td>i++</td>
<td>current = current.next;</td>
</tr>
</tbody>
</table>
Linked node problem 4

- What set of statements turns this picture:

```
list  |   data | next |
-----|-------|------|
  10 |       |      |
```

```
list  |   data | next |
-----|-------|------|
...  |       |      |
```

```
list  |   data | next |
-----|-------|------|
  990 |       |      |
```

- Into this?

```
list  |   data | next |
-----|-------|------|
  10 |       |      |
```

```
list  |   data | next |
-----|-------|------|
...  |       |      |
```

```
list  |   data | next |
-----|-------|------|
  990 |       |      |
```

```
list  |   data | next |
-----|-------|------|
 1000 |       |      |
```
Arrays vs. linked lists

- Array advantages
  - Random access: can quickly retrieve any value

- Array disadvantages
  - Adding/removing in middle is $O(n)$
  - Expanding requires creating a new array and copying elements

- Linked list advantages
  - Adding/removing in middle is $O(1)$
  - Expanding is $O(1)$ (just add a node)

- Linked list disadvantages
  - Sequential access: can't directly retrieve any value