Linked Lists

Walls and Mirrors Chapter 5
Savitch Chapter 12.1
Lewis Chapter 14.4

Preliminaries

- Arrays are not always the optimal data structure:
  - An array has fixed size – needs to be copied to expand its capacity
  - Adding in the middle of an array requires moving all subsequent elements
- ArrayLists have the same issues since they use arrays to store their data.

Objects and references

- Object variables do not actually store an object; they store the address of an object's location in the computer's memory (references / pointers).

  Example:
  ```java
  int[] values = new int[5];
  int x = 1;
  ```

Java References

- When one reference variable is assigned to another, the object is not copied; both variables refer to the same object.

  ```java
  int[] a1 = {4, 5, 2, 12, 14, 14, 9};
  int[] a2 = a1; // refers to same array as a1
  a2[0] = 7;
  System.out.println(a1[0]); // 7
  ```

Self references

- What would happen if we had a class that declared a reference to itself?

  ```java
  public class StrangeObject {
    String name;
    StrangeObject other;
  }
  ```

  Will this compile?
  - If so, what is the behavior of the other field? What can it do?
  - If not, why not?

Linking self-referential nodes

```java
public class IntegerNode {
    int item;
    IntegerNode next;
}
```
The complete IntegerNode class

```java
public class IntegerNode {
    private int item;
    private IntegerNode next;
    public IntegerNode(int item) {
        this.item = item;
        this.next = null;
    }
    public IntegerNode(int item, IntegerNode next) {
        this.item = item;
        this.next = next;
    }
    public void setNext(IntegerNode nextNode) {
        next = nextNode;
    }
    public IntegerNode getNext() {
        return next;
    }
    public Object getItem() {
        return item;
    }
    public void setItem(Object item) {
        this.item = item;
    }
}
```

Exercise: Write code to produce the following list

```
42 ——— 3 ——— 17 ——— 9
```

What set of statements turns this list:

```
lit ——— item ——— next
          10 ——— 20
```

Into this?

```
lit ——— item ——— next
          30 ——— 10 ——— 20
```

A more flexible version

```java
public class Node {
    private Object item;
    private Node next;
    public Node(Object item) {
        this.item = item;
        this.next = null;
    }
    public Node(Object item, Node next) {
        this.item = item;
        this.next = next;
    }
    public void setNext(Node nextNode) {
        next = nextNode;
    }
    public Node getNext() {
        return next;
    }
    public Object getItem() {
        return item;
    }
    public void setItem(Object item) {
        this.item = item;
    }
    Node n = new Node(new Integer(6));
}
```

Suppose we have a long chain of nodes:

```
head ——— item ——— next
          10 ——— 20 ——— ...
```

And we want to print all the items.
Printing a linked list

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

How do we walk through the nodes of the list?

head = head.next;  // is this a good idea?

Item

10

20

990

Interim summary – why should I care?

- Linked list:
  - A self referential structure
  - Objects stored – nodes
  - Advantage over arrays – no bound on capacity – can grow/shrink as needed (a dynamic structure)
  - Linked lists are the basis for a lot of data structures!
    - Stacks, queues, trees
  - The primary alternative to arrays