Recursion

Ch. 12

Recursion

- Recursion − when something is defined by itself.
- Recursive method − a method that calls itself.
- With a recursive method, we **need**:
  - A base case
  - A recursive call

Recursion

- A very simple recursive program:

```
public class RecursionEx{
   public static void main(String[] args){
      recur(5);
   }
   public static void recur(int n){
      System.out.println(n);
      if(n > 0)
         recur(n-1);
   }
}
```

Recursion

- What if we don't have a base case?

```
public class RecursionEx{
    public static void main(String[] args){
        recur(5);
    }
    public static void recur(int n){
        System.out.println(n);
        recur(n-1);
    }
}
```

```
// A simple program to compute factorials recursively
public class FactorialRecur{
   public static void main(String[] args){
      System.out.println(fact(7));
   }
   public static int fact(int n){
      if(n <= 1)
         return 1;
      else
         return n * fact(n-1);
   }
}
```

```
// Computes the nth element in fib sequence
// recursively.
public class FibSequence{
   public static void main(String[] args){
      System.out.println(fib(7));
   }
   public static int fib(int n){
      if(n <= 1)
         return 1;
      else
         return (fib(n-2) + fib(n-1));
   }
}
```

```
// Computes exponents recursively
public class FibSequence{
   public static void main(String[] args){
      System.out.println(exp(2,3));
   }
   // computes a^b
   public static int exp(int a, int b){
      if(b <= 1)
         return a;
      else
         return a * exp(a, b-1);
   }
}
```

```
// Write the method sum below that computes the sum
// of numbers between 1 and n recursively
public class FibSequence{
   public static void main(String[] args){
      int n = 6;
      System.out.println(sum(n));
   }
   // Your method goes here









}
```

Linked List

```
// The Node class is defined recursively
public class Node{
    int data;
    Node next;
    Node(int d, Node n){
        data = d;
        next = n;
    }
}
```

Simple build of a Linked List

```
public class ListTest{
    public static void main(String[] args){
        Node head = new Node(1, null);
        head.next = new Node(1, null);
        head.next.next = new Node(2, null);
        head.next.next.next = new Node(3, null);
        head.next.next.next.next = new Node(5, null);
    }
}
```

Simple build of a Linked List

```
public class ListTest{
   public static void main(String[] args){
      // we could have equivalently stated
      Node head = new Node(1, new Node(1, new Node(2,
                  new Node(3, new Node(5, null)))));
   }
}
```

Traversing a Linked List **recursively**

```
public class ListTest{
   public static void main(String[] args){
      // Assume head is pointing to the front of a
      // linked list
      traverse(head);
   }
   public static void traverse(Node n){
      if(n != null){
         System.out.println(n.data);
         traverse(n.next);
      }
   }
}
```

Stack – Linked List

Recall, stack is last in, first out. (Queue is first in, first out. LIFO v. FIFO)

- push – add item to front of list
- pop – remove item from front of list and return
- insert – insert item at index n
- find – find item at index n and return
- remove – remove item at index n