Lecture 11: Recursion (hard)

- Recap/finish: stutter an integer
  - 348 --> 334488
  - We added the base case
  - So how do we deal with larger numbers - how can we split them up?
    - We need to split into smaller chunks
    - Since our base case is 1 digit, we need to process 1 digit
    - We can do this with /10 and %10
    - Then we have 34 and 8. Let's trust that our method works; we could call it to turn these into 3344 and 88.
    - How do we combine 3344 and 88 into a single number?
    - Can't simple add --> would give 3432
    - Instead, we want 334 THOUSAND - we need to shift over the 3344!
    - Multiply by 100
  - What about negative numbers?
    - Special case for these
- Other problems:
  - Recursively sum an integer array
    - Turns out to be challenging with recursion
    - What's the base case? Empty array?
    - But then what is the recursive case? We can't create new arrays
    - Write the iterative version
    - Is there a little bit of this puzzle that we can pick off?
    - Yes, if we can refer to indexes!

      ```
      sum(entire list) = list[0] + sum(list starting at 1)
      ```
    - When would we stop in this case?
    - So we need to know which index we're at - we need another parameter
    - Public/private - sometimes this is good (want to do extra stuff beforehand - eg exception checks)

```
// returns the sum of the numbers in the given array
public int sum(int[] list) {
    return sum(list, 0);
}

// computes the sum of the list starting at the given index
private int sum(int[] list, int index) {
    if (index == list.length)
        return 0;
    else
        return list[index] + sum(list, index + 1);
}
```

- isPalindrome(String s)
  - Using a string

```
     // Returns true if the given String reads the same
     // forwards as backwards and false otherwise.
public static boolean isPalindrome(String word) {
     if (word.length() < 2) {
          return true;
     } else {
          char first = word.charAt(0);
          char last = word.charAt(word.length() - 1);

          if (first != last) {
              return false;
          }
          return isPalindrome(word.substring(1, word.length() - 1));
     }
}
```

- printBinary

```
    // writes the binary representation of n to System.out
    public static void writeBinary(int n) {
        if (n < 0) {
            System.out.print("-");
            writeBinary(-n);
        } else if (n < 2)
            System.out.print(n);
        else {
            writeBinary(n/2);
            System.out.print(n % 2);
        }
    }
```

- pow(base, exp)

```
            public static int pow(int base, int exp) {
                 if (exp == 0) {
                      return 1;
                 } else if (exp % 2 == 0) {
                      // 2^16 --> (2^2)^8
                      // base^exp -> (base^2)^exp/2
                      return pow(base * base, exp / 2);
                 } else {
                      return base * pow(base, exp - 1);
                 }
            }
```

- Crawler
  - Let's write a program that prints out the contents of a file (with indentation indicating files within folders)
    - Like the "file browser" of your operating system
  - Files are recursive structures
    - A file can be a single file
    - Or it can be a directory that contains other files
    - So it makes sense to use recursion to explore them
  - We'll prompt the user for a file and then print that file (and possibly its contents, if it has any)
    - Java has a File class (that we've used before) that will help us
    - Examine the Java API for more information
    - (show the starter code)
    - The real work will be in the print method (right now, very boring - show what it does)
  - Starter code does the right thing for files
  - We're going to work towards the final solution bit-by-bit, doing pieces and putting them together
- But we want to print out all the contents of directories
  - The Java File class represents both files AND directories
  - We can give it a File linked to a file, or a File linked to a directory
  - To do this, we need to look at all the files inside the directory
  - It turns out the File class has this ability: listFiles, which returns an array of Files
  - So we can do this:

```
public static void print(File f) {
    System.out.println(f.getName());
    File[] files = f.listFiles();
    for (int i = 0; i < f.length; i++) {
        // do something with files[i]
    }
}
```

  - But even better, use a for-each loop:

```
public static void print(File f) {
    System.out.println(f.getName());
    for (File subF : f.listFiles()) {
        // do something with subF
    }
}
```

  - (remember you can't use "f" as the for-each var name b/c "f" is the parameter)
  - And what do we want to do for each subfile?

```
System.out.println("    " + subF.getName());
```

- But if I run it again and ask for a file (e.g. Crawler.java), I get a NullPointerException
  - Why?
    - Because if there are no sub-files, listFiles() returns null, and null can't be used in a for-each loop
  - So we ONLY want to do the listFiles() if the File is a directory
  - It turns out we can use the handy "isDirectory()" method of Files

```
public static void print(File f) {
    System.out.println(f.getName());
    if (f.isDirectory()) {
        for (File subF : f.listFiles()) {
            System.out.println("    " + subF.getName());
        }
    }
}
```

- So far:
  - A decent iterative solution start to the program
  - Doesn't show EVERYTHING in the directory - no subdirectories
  - This is a very important moment in this program development - you must understand what is about to happen
  - Our code doesn't deal with directories within directories
    - Some of the subF's may be directories
  - So you might be tempted to do something like this:

```
public static void print(File f) {
    System.out.println(f.getName());
    if (f.isDirectory()) {
        for (File subF : f.listFiles()) {
            System.out.println("    " + subF.getName());
            if (subF.isDirectory()) {
                ...
            }
        }
    }
}
```

  - That's the wrong way of thinking about it, and it won't work
    - What if there are sub-sub-sub directories?
    - You don't know the depth beforehand
    - In fact, there isn't an easy iterative solution
  - Instead, think about it recursively
    - What is the method that we're writing?
    - (it prints out the name (and contents) of a file (or directory))
    - Our problem is that files and directories may be handled differently
    - But the answer is often staring you in the face
    - In the for-loop, we find ourselves wanting to print either a file or a directory
    - But we already wrote a method to do that!

        - “If only we had a method...”
        - (note that I’m not making fun - I’m just emphasizing how simple the answer can really be)

```
public static void print(File f) {
    System.out.println(f.getName());
    if (f.isDirectory()) {
        for (File subF : f.listFiles()) {
            print(subF);
        }
    }
}
```

- (run it)
- This is almost right, but doesn’t deal with any indentation
    - So deeper directories/files should be indented more
    - How can we do this?
    - We need another parameter!

```
public static void print(File f, int level) {
    for (int i = 0; i < level; i++) {
        System.out.print("    ");
    }
    System.out.println(f.getName());
    ...
}
```

    - public, private pair
    - Important to do level + 1 when recursing, not level++ (explain why)
- Sierpinski fractal example
    - Replace each triangle with 3 smaller triangles