Problem 1: Functions

In this section just implement functions - do not worry about static or classes.

(1) Given side lengths width and height, Write a function that returns the size of the smallest square that holds a rectangle with those dimensions. No math utility functions are allowed.

(2) Implement a function that compute the geometric mean of an array of floating point numbers. The geometric mean of n values is the nth root of the product of the numbers. Use Math.pow.

(3) Say you have a function sample that randomly produces a value between $0 - 10$. Write a function that repeatedly invokes sample until it obtains the number 5 and then reports the number of attempts it took to obtain that number.

(4) Say we perform a survey for the amount of water each person drinks. Each survey site submits its data as a double[] and so we have a double [][] of survey data. Write a function that, given this sort of survey data, returns the average amount of water each person drinks (note that the average of averages is not the average of values).

Problem 2: Auditing

It's good to keep track of your financial interactions so you can audit them to make sure you understand where your money is going. To start off lets represent a transaction with a class called Transaction. This class should store the source of the transaction, the destination of the transaction, the quantity of money, that was transferred, and the category for the transaction. Source/Destination are assumed to be strings and category is assumed to be a string as well. Money is in USD and represented in a double.

```
// declare the Transaction class


// Define the members of the transaction class


// Specify a constructor that takes in all of the members


// Specify getters/setters
// We'll assume that the source/destination/amount
// cannot be changed but the category can be changed.
```

Now define a Ledger class that stores a dynamic number of transactions and can add them. It should satisfy the following interface:

```
interface LedgerInterface {

    // computes the most money spent in a single transaction
    double getMaxSpent();

    // computes the total amount of money transferred in a single category
    double getTotalTransferredByCategory(String category);


    // outputs the total number of transactions where the user is
    // a source or destination
    int getTransactionCount(String user);



    // add a single transaction to the ledger
    void add(Transaction);

}

// declare the Ledger class



// Define a member for the ledger that stores a dynamic number of transactions



// implement a constructor that initializes the list storing transactions



// Implement the rest of the interface
```

Problem 3: Directory Browser

Implement a simple commandline tool that takes in a command with a single argument.

```
public abstract class FilesystemEntry {

    public String path() {...}// is implemented
    public String name() {...}// is implemented
}

public abstract class File extends FilesystemEntry {

    int size();// in bytes
}

public class Directory extends FilesystemEntry {

    public static Directory() getRoot() {...}// is implemented
    public FilesystemEntry[] getEntries() {...}// is implemented

}
```

(1) Declare the class DirectoryBrowser

(2) Define a variable for the current working directory.

(3) Implement a function called getEntry that takes in a Directory and a name. This function goes through the entries of the directory trying to find the name. If the name is found then the entry is returned. Otherwise the function throws a FileNotFoundException. Recall this is a checked exception.

(4) Implement a function called setDirectory that supports and absolute and relative paths, That is, if the path starts with a / the path is with respect to the root directory and if the path does not begin with a / it is with respect to the current working directory. The path to a directory is comprised of multiple directory names separated by /. For instance, "/usr/src/localusr" is comprised of three directory names: src, local.

For each directory name in the path can be accessed with the help of getEntries. If a path seen is not a directory or the name is not found then the function should throw a FileNotFoundException.

(5) Define a constructor that takes in an absolute path (one that is with respect to the root directory) and another one that assumes that the default working directory is the root directory (/). The first constructor does not take responsibility for any exceptions it sees.

(6) Implement listContents, which just prints out the name of each entry in the working directory.

(7) Define a function called localFileSize that adds up the sizes of the files in the working directory.

Problem 4: Cipher

A simple form of cryptography is to transform each value according to a function that we know how to invert. Your job is implement a class that uses a Cipher and then implement a Cipher yourself. Lets say we have an interface called Cipher defined by

```
public interface CharacterCipher {
    // encodes a int according to some algorithm
    public char encode(char c);
    // encodes an char according to some algorithm such that
    // c == decode(encode(c))
    public char decode(char c);
}
```

(1) Please write a class that can encode or decode an array of numbers called VectorCipher. Its constructor takes in a Cipher and it uses this Cipher to implement its encode or decode. Note that if the character is a space then that character should not be encoded.

Implement this by following most of the VectorCipher interface. Use a for loop for encode and a while loop for decode.

```
public interface VectorCipherInterface {
    // encodes every number int the input array using a specific cipher
    public int[] encode(int[] input);
    // decode an encoded int array using a specific cipher
    public int[] decode(int[] encoded);
}

// declare a class called VectorCipher




// Specify the member cipher



// Specify a constructor that takes in the member



// Implement encode using a for loop









// Implement decode using a while loop
```

(2) The Caesar cipher is one of the simplest forms of cryptography and was apparently used by Julius Caesar in his correspondences. A cipher is defined by an"offset" which specifies how many characters to "shift" an input string by. For integers this comes down to:

$$encode(d) = d + offset \tag{1}$$
$$decode(d) = d - offset. \tag{2}$$

```
// Define a Cipher called CaesarCipher


// Implement a member for the offset


// Implement a constructor for the CaesarCipher that takes in the offset




// Implement the rest of the interface
```

(3) Implement a function called encodeWord that encodes a word using the Caesar cipher with a given offset

```
String encodeWord(String str, int offset) {
    // create a CaesarCipher


    // create an array of characters to store the encoded result


    // encode each character in the string


    // return a new String based on the character array

}
```