CS206 Lab#2: Inheritance, etc

In this lab, we will design inheritance classes.

Exercise 1: Design all the necessary classes in order to make the following driver program work properly (steps have been broken down for you in the sub-parts). For example, this is a sample output that is acceptable:

```
Generally, a Dolphin can be found in water, it can not lay eggs,
and is often overheard saying 'ak, ak, ak, ak'
Generally, a Platypus can be found on land, it can lay eggs, and
is often overheard saying 'errrr'
Generally, a Human can be found on land, it can not lay eggs,
and is often overheard saying 'I'll take a grande latte with a
double-shot of espresso'
Generally, a CSStudent can be found on land, it can not lay
eggs, and is often overheard saying 'I love programming!'
```

Here is the main method for a "driver" program that might generate the above output

```
public static void main(String[] args){
  Mammal[] mammals = new Mammal[4];
  mammals[0] = new Dolphin();
  mammals[1] = new Platypus();
  mammals[2] = new Human();
  mammals[3] = new CSStudent();

  for (int i=0; i< mammals.length; i++){
    print("Generally, a " + mammals[i].getName());
    print(" can be found ");
    if(mammals[i].livesInWater() == false){
      print("on land, ");
    }
    else {
      print("in water, ");
```

Specifically, perform the following tasks. In a new project:

1. Design a class `Mammal` with:
    a. two `private String` variables called `name` and `sound`
    b. a constructor that initializes the two variables
    c. getters for the two instance variables
    d. a `void` method `speak()` that prints the object's `sound`
    e. a `boolean` method `laysEggs()`
    f. a `boolean` method `livesInWater()`
2. Design a class called `Dolphin` that extends `Mammal`. Override methods as appropriate.
3. Design a class called `Platypus` that extends `Mammal`. Override methods as appropriate.
4. Design a class called `Human` that extends `Mammal`. Override methods as appropriate.
5. Design a class called `CSStudent` that extends `Human`. Override methods as appropriate.
6. Create a `Main` class that contains the code above.
7. Each class should declared `public`, and thus be stored in a separate file that matches the class name, i.e. `Mammal.java`, `Dolphin.java`, etc.

Exercise 2: ArrayLists and reading files: The **balance sheet of a Day Trader**

Day Traders buy and sell stocks, frequently. Some will make more than 1000 trades in a day.. At the end of the day, they need to know their "position" — that is the number of shares they hold of which stocks. In this lab you will implement a system for tracking day trading positions, and printing out those positions at the end of the day. (Day traders also need to know their cash position, profit/loss etc. We are only going to deal with the number of shares they hold.)

Our hypothetical Day Trader records all of their trades in a single file that has one transaction per line. Each line is of the form:

symbol amount

where:

symbol: is a stock symbol (for example IBM)
amount: is an integer, either positive or negative (for example -400).

For example, assuming that the trader starts the day with nothing, and that the file contains:

IBM +200
MSFT +100
IBM -400

then at the end of the day their position would be:

MSFT 150
IBM −200

Yes, negative positions are allowed, in the stock market this is called “short selling”. For details, watch the movie “The Big Short”.

For this exercise write a Java program to read all of the transactions from the file
`/home/gtowell/Public206/data/lab02/bigtrades.txt`
and then print out all of the non-zero positions (negative amounts are allowed). You should read the transaction file only once, so you will need to store information in a data structure. Specifically, you should use an ArrayList.

For help in debugging, there are much smaller version of this file in:
`/home/gtowell/Public206/data/lab02/trades.txt`
`/home/gtowell/Public206/data/lab02/microtrades.txt`
microtrades.txt has the data in the example above

To complete this exercise do the following (this is a suggested set of steps, you can follow your own path instead):

1. Create a class (give it a likely name like Position, you may use any name that works for you) that has instance variables for holding a stock symbol and the number of shares. It should have accessor methods as needed.

2. Create an ArrayList that holds instances of Position. As above, assume that the Day trader starts with nothing, so the ArrayList is initially empty.

3. Read the file line by line
    With a line:
        Create an instance of Position containing the data on that line
        Search through the ArrayList of existing positions for one with the same symbol name.
            If one exists, update it with the number of shares in the just created Position.
            Otherwise add the Position to the ArrayList.

4. After reading all lines in the file, print all entries in the ArrayList that have a non-zero number of shares.

It is acceptable to do steps 2,3 and 4 within a single main function.

When you are complete turn in this page with your name and the final position of the Day Trader. You can write it by hand. If you could not complete this exercise, turn in this page with a not describing how far you got. (The lab will be on line so you can work on it later)