**Improve MoneyApp**

To save time, please skim through this entire lab before beginning.

In this lab, you will keep working on MoneyApp by adding new features. As you add these new features, you may comment out parts of main() from Lab 1 of MoneyApp after it has been checked off, if you like. Please do not remove any code from the Money class.

1. Declare an array in main() called ledger. For this lab, we will limit the array’s size to 5, but please write all supporting code so that it will work for an array of any size, i.e., use a.length in methods, for-loops rather than the numeric literal 5.

You will probably need to review some array syntax. Please Google as needed.

1. Write a static method (in the client file, outside of main()) called popLedger() to populate the array. The array should be passed to the method, but there should be no return value (do you remember why?)

Use a for-loop to populate the array with 5 money objects. Each object should be stored directly into an array element, not into a variable first and then an array element.

The loop should prompt the user to enter dollars and cents for each object. Use some kind of numbering convention when prompting your user so they can know which value they are entering. Example:

Please enter dollars for Value 1: 15

Please enter cents for Value 1: 35

Please enter dollars for Value 2: 8

Please enter cents for Value 2: 1

Etc.

Remember your user will not know what a Money object is and will count like a human, starting with 1. Remember to use a.length to make this method work for any size array.

Your values should be in some random-ish order. We will be sorting them later.

1. Write another static method called printLedger() that will print the ledger to the screen on a single line. The array declared in main() should be passed to this method. Put a space between values but no commas. (Too much trouble for this lab.) Remember to use your toString().

Example:

Here is the original ledger:

$100.01 $0.00 $0.00 $99.99 $10.19

Note: In the above example, “Here is the original ledger:” should be part of a print statement in main(). The printed ledger will be a result of a call to printLedger() which you will call from main().

1. Add an equal()method to your Money class code. It will compare two Money objects and return a Boolean, true if the objects have the same dollar and cents values. When called in main(), we will call equal()on one Money object and pass it the other: m1.equals(m2)

Here is a good header to use for this method:

public boolean equal(Money other)

It is traditional to call the name of the “other” object passed to the method other. In the method body, you will then be using code that looks like this:

this.dollars == other.dollars

To test this method in main, call it using two pairs of Money objects stored in elements of the ledger array. Perhaps like this:

ledger[0].equal(ledger[1]);

To test the equal() method, compare two objects that are the same, and two that are not the same. Use the returned boolean value and an if-statement to print the results.

This means you will need to have two values in your array that are the same. Feel free to use my examples in this lab for testing.

Example output:

$0.00 and $0.00 are the same.

$0.00 and $99.99 are not the same.

(Ideally, the if-block would be in its own static method, maybe called testEqual(), and you would pass two objects to it. You can do that if you want, but it’s not required.)

1. Write a static method called searchLedger()that will search for a given value in your ledger. The method will take as arguments the ledger array and a money object to search for. The thing we are searching for is usually called target. This would make a good variable name; its data type will be Money. This method will return a Boolean, true if the Money object with the target value has been found.

This method will require calls to the equal()method you wrote above.

Since we want the search process to stop when the object is found, you should use a while-loop. Do **not** use a for-loop and a break statement.

Test this method in main().

The value $0.00 was found.

The value $100.00 was not found.

(Ideally, we would prompt the user for the value to be found, but for this lab, you don’t have to. Unless you want to. Just finish everything else first.)

1. Write a static method to sort your array into numeric order from smallest to largest. You may use a bubble sort or a selection sort. You can find code for either via Google or in notes/labs from last semester. Please give your method a name that tells us which sorting algorithm you are using.

Your array should be passed to the sorting method. There should be no return value (do you remember why?)

Use the printLedger()method written earlier to print out the sorted array, along with a statement about what we are seeing.

1. The final output from this lab will look something like this:

Here is the original ledger:

$100.01 $0.00 $0.00 $99.99 $10.19

$0.00 and $0.00 are the same.

$0.00 and $99.99 are not the same.

The value $0.00 was found.

The value $100.00 was not found.

Here is the sorted ledger:

$0.00 $0.00 $10.99 $99.19 $100.01

When all of this is working, structurally sound, and beautiful, you will be ready for check-off.