Advancement Programming Exam
January 28, 2011

No texts, notes, or internet access are allowed on this exam. Please read these instructions **CAREFULLY** and **COMPLETELY** before starting.

Your desktop contains a folder “W11APE”. That directory contains the following items.

4 sub-folders titled: **DataAbstraction, LinkedList, Recursion**, and **General**.

Each of these folders contains a programming problem that you must address. The LinkedList folder and the Recursion folder contain subfolders. The subfolders in LinkedList are **Part1 – addFirst**, **Part2 – addLast**, **Part3 – sort**, and **Part4 – remove**. The subfolders in Recursion are **Part1 – LinkedList** and **Part2 – GCD**.

You do not need to work in any particular order, each of the folders can be completed independently from the others. If you are stuck, please free to try the other parts. NOTE: There are some .class files in the folders. ABSOLUTELY DO NOT delete or you will never get your code to compile. Leave the .class files intact.

Each folder has a file named output.txt file that illustrates the proper output your program should produce. There are also various source files and .class files in each folder. Do not modify or attempt to examine any of the .class files. Furthermore, only modify the source files you are told to for each part. Any other modifications will result in 0 points for that part.

All work you do on the exam must be done from inside this folder.

If you are unsure of anything you are being asked to do, seek help from the exam administrator.

DataAbstraction:

You are provided an abstract Stock class. You will need to write the compareTo method for the Stock class. Note: the Stock class implements Comparable<Stock>. Sort Stocks based on the number of shares and then secondarily on the symbol.

Generate two subclasses (**CommonStock.java** and **PreferredStock.java**) that extend the superclass (**Stock.java**). The subclasses contain no unique fields.

These subclasses explicitly invoke the superclass's constructor and supply the concrete method to over-ride an abstract method that is within StockInterface. The abstract method header is **public abstract double calcDividend()**. For **CommonStock** there is no dividend. For PreferredStock the dividend is 1 cent for every share.

The Stock class contains a **toString** that returns a String containing the symbol, the number of shares and the price on separate lines. You will need to write a **toString** method for **CommonStock** and **PreferredStock** that calls the toString from Stock and then adds the dividend amount on a separate line. Don’t worry about the units on the dividend amount or formatting the dividend amount.

Execution of the driver file should produce the output shown in the output file. NOTE: Do ***not*** attempt to modify **Tester.java**.

LinkedList:

There are 4 subfolders within the LinkedList Folder.

- Part 1 - addFirst: Write the **clear()**, **addFirst(),** and **toString()** methods for **LinkedList.java**. Do not attempt to modify **Tester.java**. Check the output file (**output.txt**) to see what should be produced from running the driver. **Note:** there are two fields (head and size) within the LinkedList class. The type passed into **addFirst** must be Comparable.

- Part 2 - addLast: Write the **addLast()** method for **LinkedListA.java**. Do ***not*** attempt to modify **Tester.java**. Do **NOT** delete **LinkedList$Node.class,** or **LinkedList.class** from the folder. Check the output file (**output.txt**) to see what should be produced from running the driver. The type passed into **addLast** must be Comparable.

- Part 3 - sort: Write a **sort()** method for **LinkedListB.java**. No Java API sort routine calls are allowed. Do ***not*** attempt to modify **Tester.java**. Do **NOT** delete **any of the .class files** from the folder. Check the output file (**output.txt**) to see what should be produced from running the driver.

- Part 4 - remove: Write the **remove()** method for **LinkedListc.java**. The method takes an int index as a parameter. It should remove the element at the specified position in this list. It should return the element previously at the specified position. If index is out of range then throw an **IndexOutOfBoundsException**. The first node in the list is index 0. Do **NOT** attempt to modify **Tester.java**. Do **NOT** delete **any of the .class files** from the folder. Check the output file (**output.txt**) to see what should be produced from running the driver.

Recursion:

There are 2 subfolders within the Recursion Folder

- Part 1 - LinkedList: Write the **listReverse()** method for **LinkedListD.java**. This method should print the contents of the list in REVERSE order. You must utilize recursion to accomplish this task. You are welcome to write a helper method to aid your recursion. Do **NOT** attempt to modify **Tester.java**. Do **NOT** delete **any of the .class files** from the folder. Check the output file (**output.txt**) to see what should be produced from running the driver.

- Part 2 - GCD: Write Euclid's GCD algorithm recursively. Pseudocode for the algorithm is provided at the top of the Tester file. Do **NOT** attempt to modify **Tester.java**. Do **NOT** delete **any of the .class files** from the folder. Check the output file (**output.txt**) to see what should be produced from running the driver. [For two non-negative numbers a and b, the gcd is a, when b is equal to 0; otherwise the gcd is recursively calculated with b and the result of a mod b]

General:

Modify the **Tester.java** file to:

- Connect a *Scanner* to the file name specified (**openInputFile()** method). Handle the *FileNotFoundException* exception with a try/catch and return a boolean success/failure flag.

- Connect a *PrintStream* to the file name specified: print an error message and exit the program on failure (**openOutputFile()** method). You must use a try/catch to handle the failure

- Disconnect from the input source, and disconnect from the output file (**closeInputFile()** and **closeOutputFile()** methods).

- Write a "**read()**" method to return the contents of the file as a String. The read method will read to end-of-file and then return the string. If the file is empty then return null. Don't forget to append a carriage return to each line.

- Write a "**writeLine()**" method to print a character string to the PrintStream object passed in.

Follow the directions given in the comments in **Tester.java** for more details.

Closing Notes:

- Please logoff your machine when done.
- Please return these instructions on your way out