2018-1 Computer Programming Assignment 2

You should do the assignment on your own. You are not allowed to share codes with others and/or copy codes from other resources. If you get caught, you will lose all points from this assignment.

Grading will be done in Linux environment using Java (OpenJDK) 8, identical to that inside the lab machines. Keep that in mind when writing code in other environments. Programs having any kinds of compile errors will receive neither compile nor test case points.

Do not change the format of input and output. You cannot get any points if you do not follow the output specification and print any sorts of different output from the one in this PDF file.

Write everything, including comments, in English. Koreans may incur compile errors, which will make you get no points.

Graders do not expect you to write the codes using terms and/or classes and/or methods that are not discussed in lectures and lab sessions before the deadline. If you use those, you could get minor grade deduction. If you need any guidance related to this, first find the announcement that could be posted in ETL. Then, unless you found what you want, ask to the TA.

Upload your work in ETL. You should submit only a single TAR or ZIP file containing those files:

[1] Problem 1: Warehouse.java (W in upper case!).
[2] Problem 2: Compression.java (C in upper case!).

Do not include any subdirectories or any other files inside your archive, so that we can see your source codes right after unzipping it. Graders will deduct some of your grade if you do not follow this: note that you are not going to get any grades regarding this issue sometime later.

Due of this assignment is 11PM on April 24th. No late submission is allowed.

If you have any questions, write an article to the Class Q&A board in ETL so that everyone can see what is going on. TAs will try to respond your questions and announce modifications of the specification promptly, if any. However, TAs will not be able to answer questions after 5PM on April 20th: it is kindly recommended to start your work as early as possible.

Problem 1

Write a program Warehouse.java that includes a class `Warehouse`. This class implements a retailer that (a) imports certain computer parts from their original companies and (b) sells/exchanges those parts to the customers.

<Description>

(a) common

- You need to maintain two files that contain the following entries. A line contains an entity with all those entries (separated by commas).

stock.txt: `<part id>,<part type>,<part name>,<part price>,<part quantity>`

tx.txt: `<tx id>,<tx type>,<part id>,<part price>,<tx quantity>,<tx price>`

A set of example TXT files will be given later.

- Do not destroy tx.txt if it already exists: append the content.
- ID(`<part id>`, `<tx id>`)s are numeric values which begin from 1. Those should correspond to the line number they are stored: for example, if a certain entity exists in the 5th line of a file, its ID should be 5.
- `<part name>` is a string that does not have any comma(‘,’)s.
- Prices (`<part price>`, `<tx price>`) and `<part quantity>` are numeric values with non-negative integers. However, `<tx quantity>` is a positive integer (which cannot be zero).
- There are five different types of parts (`<part type>`): CPU (C), Motherboard (M), RAM (R), GPU (G) and Power Supply (P). Values inside parentheses are the ones you need to put inside `<part type>`.
- Sequence of orders will be given as a file input, explained below.

(b) imports

- Input will be given as a single line

  `I,<part type>,<part name>,<part price>,<part quantity>`

- Update price and quantity in stock.txt and add an entity in tx.txt.
- If price of the part is changed, print either “`The part became expensive.`” or “`The part became cheap.`” in the console.
- If the part does not exist in stock.txt, you need to create an entity inside it.

(c) sells

- Input will be given as a single line

  `S,<part type>,<part name>,<tx quantity>`

- Update quantity in stock.txt and add an entity in tx.txt.
- If the part does not exist in stock.txt, or quantity exceeds the one in stock.txt, you need to exit the program with an error. Do not update both files in this case.
- You do not need to delete an entity in stock.txt even if the quantity becomes 0.

(d) exchanges

- Input will be given as a single line

  `E,<part type>,<part name>,<tx quantity>`

- Add an entity in tx.txt.
- If the part does not exist in stock.txt, you need to exit the program with an error. Do not update both files in this case.
- If quantity exceeds the one in stock.txt, print “`The part partially exchanged.`” in the console and add an entity in tx.txt.

<Example>

```
[cp00@cp2018] ~$ javac Warehouse.java
[cp00@cp2018] ~$ ls
Warehouse.class Warehouse.java input.txt stock.txt tx.txt
[cp00@cp2018] ~$ cat stock.txt
1,C,Intel i7-7700K,300000,1
2,C,AMD Ryzen 7 1700,200000,3
3,G,NVIDIA GTX1070Ti,800000,10
4,R,Samsung DDR4-17000 16G,200000,0
5,R,SK Hynix DDR4-19200 16G,240000,1
6,P,Noname 800W,80000,3
[cp00@cp2018] ~$ cat tx.txt /* Nothing yet. */
[cp00@cp2018] ~$ cat input.txt
I,C,Intel i7-7700K,300000,2
I,C,Intel i7-8700K,330000,2
I,C,AMD Ryzen 7 1700,190000,2
S,C,Intel i7-8700K,1
S,G,NVIDIA GTX1070Ti,6
E,P,Noname 800W,5
[cp00@cp2018] ~$ java Warehouse
The part became cheap.
The part partially exchanged.
[cp00@cp2018] ~$ cat stock.txt
1,C,Intel i7-7700K,300000,3
2,C,AMD Ryzen 7 1700,190000,5
3,G,NVIDIA GTX1070Ti,800000,4
4,R,Samsung DDR4-17000 16G,200000,0
5,R,SK Hynix DDR4-19200 16G,240000,1
6,P,Noname 800W,80000,3
7,C,Intel i7-8700K,330000,1
[cp00@cp2018] ~$ cat tx.txt
1,I,1,300000,2,600000
2,I,7,330000,2,660000
3,I,2,190000,2,380000
4,S,7,330000,1,330000
5,S,3,800000,6,4800000
6,E,6,80000,3,240000
[cp00@cp2018] ~$
```

Problem 2

Write a program Compression.java that includes a class `Compression`. This class implements two different encoding schemes: (a) Huffman coding (https://en.wikipedia.org/wiki/Huffman_coding) and (b) Tunstall coding (https://en.wikipedia.org/wiki/Tunstall_coding).

<Description>

- You need to implement both compression and decompression inside the `Compression` class.
- You do not need to understand the mathematical details of how dictionary/codeword is constructed: you can simply use the dictionary files to compress and decompress, as explained below.
- A dictionary dictionary.txt contains a sequence of pairs of the form:

`<string>,<codeword>`

. In case of Huffman dictionary, `<string>` consists of an alphabet (single character), whereas in the case of Tunstall, `<string>` represents a sequence of alphabets. `<codeword>` is a binary sequence (i.e., a sequence of '0's and '1's).

- The input file, input.txt contains:

`<algorithm type>,<algorithm mode>,<string>`

. `<algorithm type>` is either H (Huffman) or T (Tunstall). `<algorithm mode>` is either C (compression) or D (decompression). `<string>` is the string to be compressed or decompressed.

- Result of the operation should be saved into the file output.txt.

<Example>

[cp00@cp2018] ~$ `javac Compression.java`
[cp00@cp2018] ~$ `ls`
Compression.class Compression.java dictionary.txt input.txt
[cp00@cp2018] ~$ `cat dictionary.txt`
A,010
B,011
C,11
D,00
E,10
[cp00@cp2018] ~$ `cat input.txt`
H,C,AEBACBEDDA
[cp00@cp2018] ~$ `java Compression` /* Huffman compression. */
[cp00@cp2018] ~$ `cat output.txt`
01010011010110111100000010
[cp00@cp2018] ~$ `ls`
Compression.class Compression.java dictionary.txt input.txt output.txt
/* A new input.txt file is added. */
[cp00@cp2018] ~$ `cat dictionary.txt`
A,010
B,011
C,11
D,00
E,10
[cp00@cp2018] ~$ `cat input.txt`
H,D,01010011010110111100000010
[cp00@cp2018] ~$ `java Compression` /* Huffman decompression. */
[cp00@cp2018] ~$ `cat output.txt`
AEBACBEDDA
/* The example continues on the next page. */

[cp00@cp2018] ~$ `ls`
Compression.class Compression.java dictionary.txt input.txt output.txt
/* A new dictionary.txt and input.txt files are added. */
[cp00@cp2018] ~$ `cat dictionary.txt`
A,000
BA,001
BB,010
BC,011
BD,100
C,101
D,110
[cp00@cp2018] ~$ `cat input.txt`
T,C,ADCABCADCBA
[cp00@cp2018] ~$ `java Compression` /* Tunstall compression. */
[cp00@cp2018] ~$ `cat output.txt`
00011010100001100011010101001
[cp00@cp2018] ~$ `ls`
Compression.class Compression.java dictionary.txt input.txt output.txt
/* A new input.txt file is added. */
[cp00@cp2018] ~$ `cat input.txt`
T,D,00011010100001100011010101001
[cp00@cp2018] ~$ `java Compression` /* Tunstall decompression. */
[cp00@cp2018] ~$ `cat output.txt`
ADCABCADCBA
[cp00@cp2018] ~$