Data Structures (CS 102), Professor Yap – Fall 2016

HOMEWORK SOLUTION

September 20, 2016

Homework 1: Due on Thursday Sep 15, 2016

- The main purpose of this homework is to review some basic Java concepts, learn to use random numbers and do file I/O. We also test your close reading of the assigned Chapter 1.
- This homework is worth 100+5 Points. In each homework, you get the extra 5 points by following all our instructions. Typically, our homework has two parts: a written part and a programming part.
- Reference [DJW-book] is our textbook of Dale, Joyce and Weems. Our reading assignment (See Class Wiki) of Chapter 1 is tested here. Exercises in [DJW-book] are cited as `Ex.XXX.YYY` where `XXX` is the chapter and `YYY` is the exercise number.
- The Programming Page in our Class Wiki may get you started with Java and Eclipse, and useful as a reference. We strongly recommend learning to use some kind of text editor such as `gvim`.
- READ CAREFULLY the homework instructions and policy in `http://cs.nyu.edu/~yap/wiki/class/index.php/DataStruc/Hw-page`. Moreover, you must submit only a single zip file called `hw1_YYY.zip` where `YYY` is your last name (with only the first letter in caps). E.g., I would submit `hw1_Yap.zip`. Please download the sample zip file by this name in Piazza/Resources/Homework. You may modify the `README` file there for your own submission.

PART A: WRITTEN ASSIGNMENT

Question A.1 (1 Points) Ex.1.1 (software engineering)

SOLUTION: Software engineering is a disciplined approach to the creation and maintenance of computer programs.

Question A.2 (4 Points) Ex.1.7 (modifying software)

SOLUTION:
a. When the program's requirements change; when a better solution is discovered in the middle of the design phase; when an error is discovered in the requirements due to the design effort.
b. When the program is being debugged, because of compilation errors or errors in the design; when a better solution is found for a part of the program that was already implemented; or when any of the situations in Part (a) occur.
c. When there are errors that cause the program to crash or to produce wrong answers; or when any of the situations in Parts (a) or (b) occur.
d. When an error is discovered during the use of the program; when additional functions are added to an existing software system; when a program is being modified to use on another computer system; or when any of the situations in Parts (a), (b), or (c) occur.

Question A.3 (7 Points) Ex.1.28 (legal or not)

SOLUTION:
a. Legal - getDay is a public method that returns an int
b. Legal- getYear is a public method that returns an int
c. Illegal - increment is not defined for Date objects
d. Legal - increment is defined for IncDate objects
e. Legal - object variables can be assigned to objects of the same class
f. Legal - subclasses are assignment compatible with the superclasses above them in the class hierarchy
g. Illegal - superclasses are not assignment compatible with the subclasses below them in the class hierarchy

Question A.4 (3 Points) Ex.1.42 (equal or not)

SOLUTION:
Not equal - since date1 and date2 refer to different instances.
Equal - since data1 and date2 now refer to the same instance.
Equal - since data1 and date2 still refer to the same instance which happened to be updated.

Question A.5 (12 Points) Ex.1.46 (order of magnitude)

SOLUTION:
a. $O(N^2)$ - since $3N = O(N^2)$ and $O(N^2) + O(N^2) = O(N^2)$
b. $O(N^2)$ - for the same reason as part a.
c. $O(N^5)$ - since each of the 3 terms are $O(N^5)$, and $O(N^5) + O(N^5) + O(N^5) = O(N^)$.
d. $O(N^2)$ - using the fact that $3N \log N = O(N^2)$, we conclude that $O(N^2) + O(N^2) = O(N^2)$.
e. $O(N^4)$ - since each of the 5 terms are $O(N^4)$, and so their sum is also $O(N^4)$.
f. $O(N^2)$ - since we can expand the expression into $N^2/2 - N/2$, and thus $N^2/2 - N/2 \leq N^2/2 = O(N^2)$.

Question A.6 (5 Points) Ex.1.48 (big-Oh)

SOLUTION:
a. $O(N)$ - the running time is $O(1) + O(N) = O(N)$
b. $O(N^2)$ - the running time is $O(1) + O(N^2) = O(N^2)$
c. $O(\log N)$ - each iteration we half the value of N. So in k steps, the value is $N/2^k$. So when $2^k > N$, the value is less than 1 and we stop, i.e., $N/2^k < 1$. That means that at the previous step (i.e., at step $k-1$), we have $N/2^{k-1} \geq 1$. Taking log base 2 of both side, we get $\log_2 N - (k-1) \geq 0$. Thus, $(k-1) \leq \log_2 N$, or $k \leq 1 + \log_2 N \leq 2\log_2 N$. I.e., the number of steps k is at most $2\log_2 N = O(\log N)$.
d. $O(1)$ - since there are four steps.
e. $O(N)$ - since the running time is $1 + O(N) + 1 + O(N) = O(N)$.

Question A.7 (10 Points) The following program is meant to print the message “I am Homework 1”.

```
public class hw1A {
    public void show(){
        System.out.println(”I am Homework 1”);
    }
    public static void main (String[ ] args){
        show();
    }
}
```

Unfortunately, the code has a compiler error.

(a) What is the error message? Explain what happened.

(b) Give two distinct fixes for the problem. (Write the Java code, and describe in words how it solves the problem.)

SOLUTION: (a) Error Message:
`hw1A.java:13: error: non-static method show() cannot be referenced from a static context.`
That is because `main` is a static method, while `show` is an instance method. Static methods have no access to instance methods or instance variables.
(b) A very simple solution is to make `show` a static method. Replace its current signature by the following:

```
static void show()
```

This solution simply changes ONE key word in the original program. Alternatively, leave the `show` method intact, but modify the body of the `main` method to the following:

```
hw1A s = new hw1A();
s.show();
```

In other words, we have to first create an instance of the `hw1A` class, and use this instance to access the `show()` method!
A third solution is to move the `show()` method into a new class, say `showClass`. The class `showClass` has just one method, namely `show()`. Note that `show()` can be static or public, but if it can be static, then you might as well use our first solution.
Comments: Which of these solutions preserves the intent of the original programmer best? The simplest fix (making `show()` a static method) might be OK here, but we can imagine an elaboration in which `show()` actually access instance variables. In that case, this solution is NOT an option. But the other solutions will still work.

Question A.8 (3 Points) Odd One Out
One of the following words does not belong in the list:

reference, link, address, null, pointer, alias

How are all these words (with one exception) similar? Read Text p.34.

SOLUTION: “null” is the odd one out. All the other words are roughly synonyms!

Question A.9 (5 Points) Garbage
Write the smallest complete program you can compile (literally as you write it) so that will produce garbage while it is running. Be sure to say what is the garbage produced.

SOLUTION: You must have a variable that is non-primitive type. The simplest is to have an array of primitive type (say array on ints):

```
class Garbage {
    public static void main(String[ ] args){
        int[ ] g = new int[1];
        g=null; // this produced garbage!
    }
}
```

After we allocated memory for this array `g` on the stack (by calling `new`), we abandon that memory be pointing the array `g` to `null`.

PART B: PROGRAMMING ASSIGNMENT (40 Points)

Question B.1 When you unzip the sample file `hw1_Yap.zip`, you will find 4 files under the `src` folder: `HelloClass.java`, `FileHandling.java`, `Account.java`, and `students.txt`. Compile and run the Java files, and study our comments in them to understand what the code is doing.

Question B.2 (20 Points) Hello Students!
Write a program called `HiClass.java` which is a modification of `HelloClass.java` so that it reads the list of students from the text file `students.txt`. For each student, send a random greeting from the array of greetings. Print these student greetings, one line per greeting, into an output file called `hiStudents.txt`.

NOTE: feel free to add to our list of possible greetings.

Question B.3 (30 Points) Bank ATM
We want to simulate the operations of a Bank ATM Machine. Write a `ATM class` in Java, using our provided `Account class`. You must not modify the `Account.java` file in anyway. The `ATM` sits in a loop, taking commands from the users.

These are the possible commands:

(1) **Open SSS:** This opens a new account for a customer named SSS and prints #NNN where NNN is a new (automatically generated) account number. The customer SSS must remember this NNN.

(2) **Quit:** Logout of the current account. Print "Goodbye SSS!" to the user SSS.

(3) **Login NNN:** where NNN is an account number. Print "Hello SSS!" where SSS is the name of the customer.

(4) **Deposit DDD:** where DDD is the dollar (assume only integer values) amount. Print +DDD as a response.

(5) **Withdraw DDD:** Print −DDD as a response.

(6) **Balance:** Prints =DDD if the balance is DDD.

(7) **Terminate:** Print "ATM shutting down!" and exit the program. Before exitting, ATM must store all account information in a text file called `bankAccounts.txt`. This file would be read when the program is restarted.

Optionally, the user may type only the first letter of each command (O, Q, L, D, W, B, T) instead of Open, Quit, etc. To read commands from the terminal, please use the `java.util.Scanner` class (you can also study the model program in p.19 of Text). You need to keep track of all the open accounts in some collection – we recommend using the `java.util.ArrayList` class (Appendix E of Text, p.753).

When you start up the `ATM` program, the first action of the ATM is to read all the information in `bankAccounts.txt` (if this file exists). We suggest each line of `bankAccounts.txt` stores the data for a single account.

Example: Column 1 is an example of a sequence of inputs. Column 2 shows the corresponding output. `InputFile.txt`:

Command	**Output**	**Comments**
Open Chee	#1000	New account number is 1000
Login 1000	Hello Chee!	Now Chee is logged in
Deposit 100	+100	We print the amount deposited
D 50	+50	D means Deposit; print the deposit amount
Balance	=150	This is just 100+50.
Quit	Goodbye Chee!	You are logged out.
Open Fred	#1001	New account number is 1001
L 1001	Hello Fred!	Logged into account number 1001
Deposit 999	+999	Initial deposit is 999 dollars
L 1000	Goodbye Fred! Hello Chee!	Log out Fred and log in Chee (see discussion below)
W 5	−5	W means Withdraw. Print the withdrawn amount
B	=145	Shows the new balance of 145 dollars
Q	Goodbye Chee!	Prints the Goodbye Message
B	ERROR!	Cannot get balance since you are not logged in
L 1001	Hello Fred!	Logged into account number 1001
W 99	−99	Withdraws 99 dollars
B	=900	Shows the balance of 900 dollars
T	ATM shutting down!	Print shut down message

IMPORTANT: Make sure that your program produces EXACTLY the output shown in Column 2, one output per line.

Errors Handling: if you receive a unexpected command, you can always print “ERROR!” and continue to accept new commands. *Do not hang-up, crash, or terminate the program.* It is also important to realize that *we do not tell you what to do under all error conditions – just make some reasonable choices.*

E.g., in the above example, the command `L 1000` was issued before we logged out of Account #1001. We could regard this as an ERROR, but we choose to simply log out of #1001 and log into #1000.

E.g., what if the user types “Oven” instead of “Open”? It is OK if you just regard it as “Open” because you only check the first letter. But it is also OK print an ERROR.

E.g., what if you try to withdraw more than your current balance? It is up to you to call this an ERROR or not. If it is not ERROR, it means you are treating this as a “Checking Plus Account” which can have negative balance!

Because of all these possible scenarios, we ask that you write a short paragraph discussing your choices in these under-specified situations.

The Input Model: The concept of error is greatly affected by the input model. It is important to understand that we expect you to process the inputs on a line-by-line basis, not token-by-token basis. Typically, a line will have one or two tokens. E.g., you want to deposit $100. The command line should be ”Deposit 100”. You must not split this into two lines: ”Deposit” and ”100”. Similarly, you cannot combine two or more commands into one line: ”Deposit 100 Withdraw 15 Quit” is a problem. But here, we want you to ignore the rest of the line but do not print ”ERROR!”.

SOLUTION: We uploaded a zip file hw1_Sol.zip in Homework Solutions of Piazza Resources. This has the written program as well as programming part. Please compile and run this solution (using the included Makefile)
Please study our solutions for the basic model: note that we use Scanner for input, and FileWriter for output. Please become familiar with the methods of these two classes.
You can use the sample `Input.txt` file by using redirecting the standard input `stdin` with the shell operator "< `src/Input.txt`" (recall Lecture 3). Do it by hand, but the Makefile can achieve this as well at command line via `ifile=src/Input.txt`.