The State Universtiy of New York, Korea
Computer Science

CSE 114 Handout 1: Problem Set 1 March 6, 2019

Read This Right Away

This problem set is due at **11:55pm on Wednesday, March 13, 2019, KST**. Don't go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

- To solve each problem below, you will be implementing a Java class.
- Please read carefully and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
- You should create your programs using emacs.
- Your programs should be formatted in a way thats readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
- This problem set assumes you have installed Java and emacs on your computer. Please see the course web for installation instructions.
- You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this problem set. Do not use Eclipse yet.
- **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied in class so far.

What to Submit

Submit the following files on **Blackboard**. Multiple submissions are allowed before the due date. Please do **not** submit `.class` files or any that I did not ask for.

```
Poem.java
P2.txt
Earning.java
Ave.java
Time.java
Exchange.java
Change.java
```

Partial vs. Complete Solutions

Your programs should compile and run without errors, both compile-time and run-time errors! Please do not submit programs that do *not* even compile! Its better to submit a partial solution that compiles and runs as opposed to an almost complete solution that does not even compile. A program that does not compile even if it is very close to being perfect would only receive 30% maximum of the possible score for the program! This policy applies not only to this problem set but also to all the other ones in the remainder of the semester. I will not repeat this in each problem set though. So, please remember this.

Naming Conventions In Java And Programming Style In General

Please use these conventions as you establish your *programming style.* Programming professionals use these, too.

- **Names:** Choose informative names, e.g., `hourlyRate` rather than `hr` if it is to be a variable name; `CheckingAccount` rather than `CA` if it is to be a Java class name.
- **Class names:** We start a class name with an uppercase letter as I have in my examples: `Hello` not `hello` or `HELLO`. For multi-word names do as I do here: `UndergraduateStudent`, `GraduateStudent`, `CheckingAccount`, etc.
- **Variable names:** We start a variable name with a lowercase letter, e.g., `count`, `hourlyRate`, etc.
- **Method names:** Naming convention for method names is the same as that for variable names.
- **Use of white space:** Adopt a good indentation style using white spaces and be consistent throughout to make your program more readable. Emacs will do the indentation automatically for you. If it does not, your emacs installation was not done correctly. If you need help, see me.

I will not repeat this in each problem set, but you should follow this throughout the semester (and beyond, I suggest.).

Problem 1

Create a class named `Poem` in a file named `Poem.java`.

You will be writing a poem and it must be at least five lines long.

Hand in `Poem.java`.

Problem 2

Create a file named `Debug.java` with the following program as its content.

```
public class Debug {

    public static void main (String[] args) {
        System.out.println("I think this is gonna be fun!!!");
    }

}
```

Do the following using command line interface. Introduce the following errors, one at a time, to the program. Try to make sense out of any error message that the compiler generates. Fix the previous error each time before you introduce a new one. If no error messages are produced, explain why to yourself. Try to predict what will happen before you make each change. Of course, you will have to use 'javac' and 'java' to see any error. In other words, you will not see any error while you are editing your program in an editor such as emacs.

1. Change `Debug` to `debug` and run the program to see what happens.

2. Change `fun` to `FUN` this time.
3. Remove the first quotation mark in the string. We haven't talked about what a string is, but you may guess that it is something that is enclosed by two matching quotation marks.
4. Add a few more blank spaces between `think` and `this`.
5. Add a few more blank spaces (or tab characters) before `System`.
6. Add a few blank lines just above (or below) the line that starts with `System.`
7. Remove the last quotation mark in the string.
8. Change one of the double quotation marks with a single quotation mark.
9. Change `println` to `bogus`.
10. Remove the semicolon at the end of the `println` statement.
11. Remove one of the open squiggly-braces (') in the program.
12. Remove one of the close squiggly-braces (') in the program.
13. Remove `System.`.
14. Remove one of the parentheses. And also add an extra one.
15. Add the following two lines right below the `System.out....` line and run it once.

```
System.out.println("It would have to be.");
System.out.println("Otherwise, Art Lee is a liar!");
```

    Now, run it again after you replace each occurrence of `println` by `print` and see what happens.

For this problem you only have to hand in a file named `P2.txt` which is a *plain text file* that contains one of the following sentences as its content depending on whether you did them all or not.

```
Yes, I have done all.
```

or

```
No, I have not done all.
```

A plain text file is a file that we should be able to view using a simple editor such as emacs without requiring a special program such as MS Word or Excel or Mac TextEdit or Pages.

Problem 3

Write a Java class named `Earning` in a file named `Earning.java` that computes the amount of earning as follows. Declare three variables: one for the number of hours worked for each week, another for the hourly rate, and the third for the amount earned to be computed. Declare each of the variables as an `int`. Output the earning for a month assuming that there are four weeks in a month.

Assign a realistic number (the kind of numbers that you would see for your part time job) to each of the first two variables and compute the earning and store the computed value to the third variable.

Now, print out the three values to the standard output device, i.e., the screen, with some annotations so that the printed output is meaningful to the user.

In this problem you are not reading anything from the keyboard. You are assigning a value to a variable if needed at the time you write the code, i.e., you are using a "hard-wired" value. In other words you are assigning a value to a variable at compile time.

Hand in `Earning.java`.

Problem 4

Write a Java class named `Ave` in a file named `Ave.java` that does the following. It has five integer variables declared with the names `n1, n2, n3, n4,` and `n5`. Your program should assign an integer to each of them at compile time. The numbers that you use are up to you. Now, your program must calculate their average and store it to a variable named `average` of type `double`, and print it as output to the standard output device. The result should be a floating point number, namely a real number. A floating point number in Java can be represented as the type `float` or the type `double`. Not only in this problem but in general for the rest of this semester let us use `double` unless I specifically ask you to use `float` when we want to represent a real number. Make sure that the computed average keeps its fractional part without losing any precision. For example, if the actual average is 12.36, it should print 12.36, not 12.0.

Hand in `Ave.java`.

Problem 5

Write a Java class named `Height` in a file named `Height.java` that adds two height values as follows. For example, you can add two height values 5 feet 3 inches as one and 4 feet 11 inches as another to compute the sum as 10 feet 2 inches.

Use three sets of feet and inches variables to represent three height values. Your program should add the first two height values and store the result into the third height value.

Finally, print the three height values with some annotations so that the result will be meaningful to the user.

If you need to introduce additional variables, feel free to do so.

The first two sets of variables will get their values at compile time.

Hand in `Height.java`.

Problem 6

Write a Java class named `Exchange` in a file named `Exchange.java` that reads a dollar value from the user (namely from the keyboard) as a floating point number into a variable named `dollar`, converts it into Euro as well as Korean Won, stores them in two different variables named `euro` and `won` respectively, and prints the results to the user in a friendly format. The number of digits below the decimal point in your output can be whatever the computer gives you.

By *reading* a number I mean that your program should instruct the user of your program to enter a number from the keyboard and read it into a variable (namely assigning a value to the variable at run time), rather than assigning a number yourself to a variable at compile time.

Hand in `Exchange.java`.

Problem 7

Write a Java class named `Change` in a file named `Change.java` that reads from the keyboard a real number representing a monetary amount (a dollar value). Then, determine the fewest number of each bill and coin needed to represent that amount, starting with the highest (assume that a twenty dollar bill is the largest bill available). For example, if the value entered is 59.78 (fifty-nine dollars and seventy eight cents), then the program should print the equivalent amount as:

```
2 twenty dollar bills
1 ten dollar bills
1 five dollar bills
4 one dollar bills
```

```
3 quarters
0 dimes
0 nickles
3 pennies
```

Note that each line in the output above starts with 5 blank space characters, and your output should be in that form. Hand in `Change.java`.

Hint: Convert the amount that you read in, which is a real number, into an `int`, i.e., cents. Also if your program is correct but seems to give a result that is off by one penny, don't worry about it.