The State Universtiy of New York, Korea
Computer Science

CSE 114 **Handout 3: Problem Set 2** **March 14, 2019**

This problem set is due at **11:55pm on Thursday, March 21, 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.

Please follow the following instructions carefully. These will apply to the future problem sets as well.

- 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**.

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.

```
P123.txt
Debug.java
Ugly.java
Param.java
MultAdd.java
Date.java
```

What Java Features to Use

For this assignment, you are not allowed to use more advanced features than what we have covered in Lectures 1 through 6 (March 13 lecture) [Chapters 1 through 5 of our textbook].

Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

A Piece of Advice

This is what I said in the FAQ for a problem set around this time, when I taught this class previously. It might be helpful for you to read.

> I can see what you are doing.
>
> You are in a hurry to finish these programs. So, you try to write the whole program that would solve the entire problem quickly.
>
> Think again. Remember I have been telling you to solve them incrementally? Try a partial solution that works and little by little improve it toward the ultimate solution.
>
> I see these long programs that you wrote that is not working and you ask me to take a look and tell you why they are not working. There are so many problems with these programs that it's hard to know where to begin.
>
> Waiting until the last minute or even a few days before the due date to solve these problems isn't a good strategy and problems in the future assignments will not get any easier.
>
> Those who are on track seem to be enjoying the class. We are just getting into the phase where we start experiencing some exciting programming. Try to take advantage and enjoy!

Problem 1 (21 points)

1. (3 pts) What is the decimal equivalent of the binary number 1101001?
2. (3 pts) What is the binary equivalent of the decimal number 572?
3. (3 pts) Given 5 bits, list all of the distinct combinations of the 5 bits.
4. (3 pts) Next to each combination in your answer to the previous question, write the decimal equivalent to the binary bit pattern.
5. (3 pts) If you have n bits, how many distinct combinations of the n bits could you enumerate?
6. (3 pts) The type `int` in Java is represented as a signed integer with 32 bits of memory. What is the largest positive integer that can be expressed using the `int` type in Java? Give your answer as a binary number as well as a decimal number. Assume that Java uses the most significant bit as the sign bit.
7. (3 pts) The type `double` in Java is represented with 64 bits of memory. Let's not worry about the IEEE floating point format that Java actually uses. Instead we will cook up a simpler scheme with a similar concept as follows. With one bit, the most significant bit, used to represent the sign we have 63 bits to represent the magnitude of the number. But, we now have to divide those 63 bits into two parts: one part to represent the whole number part and the other for the digits below the decimal point – the fractional part. Let's assume that we use 30 of the remaining 63 bits to represent the whole part and the remaining 33 bits for the digits below the decimal point. What would be the largest real number that we can use in Java if we were to use this scheme to represent real numbers. Give your answer as a decimal number.

Include your answers in a plain text file named `P123.txt` and hand it in.

Problem 2 (10 points)

1. Assume that there are 485 students in a class. If every student is to be assigned a unique bit pattern as his or her student ID, what would be the minimum number of bits required to do this type of encoding?
2. How many more students can be admitted to the class without requiring any additional bits using the same encoding scheme?

Include your answers in the same file: `P123.txt`.

Problem 3 (10 points)

(a) In class we studied how information is represented in memory using ASCII code. (Java uses Unicode, but we will use ASCII code for this problem.) What would it look like in memory for the string ”Boy, CS is Fun!” (including the exclamation point but not including the quotation marks) if we used ASCII code in binary? Give one with separating blank spaces and another one without them as we did in class.

(b) Given a bit pattern, one like the answer that you obtained in part (a) above, how would you recognize that it represents the string ”Boy, CS is Fun!”? Describe in English how you might do it, namely your algorithm.

Include your answers in the same file: `P123.txt`.

Problem 4 (10 points)

This problem is designed to demonstrate to you that using `print` function is very helpful when you debug your program.

Download the file named `Debug.java` from the `[given]` link next to the problem set handout and follow the instructions step by step to see how they are used to debug a program that runs fine but does not produce the result that you expect to get.

1. Try to compile and run `Debug.java`. You will run into (a) compile-time error(s). Fix it/them and make the program free of compile-time errors. When you fix the compile-time error(s), the program will run fine and produce a result. The problem at that point is that the result produced will be wrong! We will try to locate those logical errors (yes, there are more than one) by using a debugging technique, namely, using `print` (or `println`) statements, e.g., `System.out.println(...)` as debugging statements, and fix them.

2. First, add the following line where ”Line 1” is and make sure the values that get printed are indeed the values that you expected.

   ```
   System.out.println("x1=" + x1 + " y1=" + y1 + " x2=" + x2 + " y2=" + y2);
   ```

   If the values are not what you expected, then fix the error(s).

3. This time add the following line where ”Line 2” is and make sure the value that gets printed is indeed the value that you expected.

   ```
   System.out.println("deltax = " + deltax);
   ```

   If not, then fix it.

4. This time add the following line where ”Line 3” is and make sure the value that gets printed is indeed the value that you expected.

   ```
   System.out.println("sum = " + sum);
   ```

If not, then fix it. Fixing it may require introducing some additional `System.out.println(...)` statements somewhere inside the function `sumOfSquares`. You may be able to spot the error without even typing the debugging statement(s), but I want you to add the debugging statement(s) even in that case. The point of this exercise is to learn how to debug, not just to find the right answer. It is a simple program and I am not concerned about the outcome as much as learning a debugging technique so that you will be able to apply it when you face a much more complicated situation later on.

Now that I have shown you a technique on how to use these debugging statements, use them whenever and wherever they may be helpful to debug whatever part of your program that you will be developing in later problems.

5. Fix all the bugs until your `main` produces the correct output.

Hand in the fully debugged version of `Debug.java` with the debugging statements that you added left in.

Problem 5 (16 points)

Download the file named `Ugly.java` from the `[given]` link next to the problem set.

Your task for this problem is to improve it by following the specification here:

- (2 pts) The program as it is written is terribly formatted, i.e., indentation is all off. Yes, the program indeed looks ugly. Before you do anything, make things line up nicely with the right amount of indentation in each line of the program in such a way that the structure of the program is reflected by how the code is indented. The examples I create for my lectures are all nicely indented. For example `Debug.java` in the previous problem is nicely indented. The amount of indentation I (and many professionals) use is four blank spaces. I suggest that you use the same amount. A tab will give you more than four and it will make your program harder to read when your program gets large with many levels of indentations.

  From now on badly indented programs will lose points when they are graded.

- (5 pts) Define a method (aka function) named `c2f` with one formal parameter of type `double` that returns a value of type `double`, and include it in the same class, i.e., `Ugly.java`. `c2f` converts a Celsius temperature into Fahrenheit. Note that you have just *defined* a function, but it is not being *used (applied/invoked/called)* yet. You will use it in the next step below. Remember that we define a function as the first step and then use/call/invoke/apply it afterwards? Oh, add the following as the first line of `c2f`:

  ```
  System.out.print("In c2f. . .");
  ```

- Compile `Ugly.java` to make sure that the definition of `c2f` does not have any compile-time error. We just defined a function and compiled and it compiled okay, I hope by now. So, we know at least the program so far does not have any compile-time error in it. If you run your program (i.e., `main`), it will run just as before you added `c2f` since `c2f` is not even being *called* by any function which means that it is not active yet at run-time.

- (2 pts) Now rewrite `'Line 1'` in `Ugly.java` so that it uses/calls `c2f` instead of computing it in main without using `c2f`. Compile the file and run `main` to make sure that the definition of `c2f` works at run-time as well with correct results.

- Now rewrite `'Line 2'` in `Ugly.java` so that it also uses `c2f`. Compile the file and run `main` to make sure that the definition of `c2f` works there as well. Remember that `c2f` converts a Celsius temperature into Fahrenheit and that is exactly what we need here.

- (5 pts) Define a method (function) named `f2c` with one formal parameter of type `double` that returns a value of type `double`, and introduce it into the same class, i.e., `Ugly.java`.

- (2 pts) Now rewrite `'Line 3'` in `Ugly.java` so that it uses/calls `f2c`. Compile the file and run `main` to make sure that the definition of `f2c` works without any syntactic (compile-time) or logical (run-time) error.

- Now rewrite `'Line 4'` in `Ugly.java` so that it uses/calls `f2c`. Compile the file and run `main` to make sure that the definition of `f2c` works there as well.

Hand in your revised `Ugly.java`.

Problem 6 (10 points)

The point of this exercise is to help you understand how to write and invoke methods that take parameters. Create a class named `Param` in a file named `Param.java` and do the following:

- Introduce a method named `pars` that takes five formal parameters: an `int`, a `double`, another `double`, a `boolean`, and a `String` in that order. This method then prints those values to the standard output device (screen) one on each line with some annotations that describe what your function is printing. This method does not return anything useful, which is another way of saying the return type of the function is to be `void`.

- Introduce a `main` method that establishes five values that would be needed to call `pars` and actually calls it. The four values that you will use are the year of your birth day, the current 30-year fixed mortgage rate (obtained from any credible source), the current currency exchange rate in euros for a dollar, whether the statement (Today is Sunday.) is true or not, and the name of your home state or country.

  These values are to be assigned at compile-time. That is, they will be hard-coded. In other words, you will not be reading the values from keyboard at run time. Unless I specifically asked that you read input from keyboard, you are not expected to do so here as well as in other problems in this problem set and future problem sets.

- Compile your program and run it to make sure that it works without any compile-time or run-time errors.

Hand in `Param.java`.

Problem 7 (20 points)

Many computation can be expressed concisely using the “multAdd” operation, which takes three operands and computes $a * b + c$. Some processors even provide a hardware implementation of this operation for floating-point numbers.

- Create a class named `MultAdd` in a file named `MultAdd.java`.

- Write a method named `multAdd` that takes three `double`s as parameters and that returns their multadditionization.

- Write a `main` method that tests `multAdd` by invoking it with a few simple parameter values, like `1.0, 2.0, 3.0`, and then prints the result, which should be `5.0`.

- Also in `main`, use `multAdd` to compute the following values, calling it once for each and printing the computed result each time in `main`:

$$cos\frac{\pi}{4} + \frac{sin\frac{\pi}{4}}{3}$$

$$log10 + log20$$

  You may use any log function, i.e., natural log or base 10 log.

- Write a method called `yikes` that takes a `double` as a parameter and uses `multAdd` to calculate and print the result of the following formula:

$$xe^{-x} + \sqrt{1 - e^{-x}}$$

  Add a piece of code in your `main` to test your implementation of `yikes`.

Note that `yikes` does not return anything useful. It generates the computed value as output to the standard output device. You may wonder why we make `multAdd` return the computed value whereas we make `yikes` print the computed value to the output device rather than return it to the caller. It is really up to the programmer's design. I just wanted you to learn two different ways of doing these things. HINT: the Math method for raising e to a power is `Math.exp`. You will also find `Math.sqrt` useful.

In the last part, you get a chance to write a method (first one, i.e., `yikes`) that invokes a method that you already wrote (second one, i.e., `multAdd`). Whenever you do that, it is a good idea to test the second method carefully before you start working on the first. Otherwise, you might find yourself debugging two methods at the same time, which could get quite complicated if you are dealing with complex methods.

One of the purposes of this exercise is to practice pattern-matching: the ability to recognize a specific problem as an instance of a general category of problems.

Hand in a fully debugged and tested version of `MultAdd.java`.

Problem 8 (10 points)

Implement a class named `Date` in a file named `Date.java`. Implement the following function in the class:

```
public static String alphaToNum (String m, int d, int y);
```

where `m` is for month, `d` for day, and `y` for year. You may assume that you only get valid actual arguments. That is, you do not have to worry about handling error situations due to invalid input values. You may assume that values will be of correct type and in a valid range. Below is an example of a call to the method and its return value.

```
alphaToNum("March", 14, 2019) = "3/14/2019"
```

Include at least three calls to `alphaToNum` in your `main` to test your implementation. Here is an example of `main` calling it once. Note how I used `print` and `println`.

```
public static void main(String[] args) {
    System.out.print("alphaToNum(\"March\", 14, 2019) should be 3/14/2019");
    System.out.println(" : " + alphaToNum("March", 14, 2019));
}
```

Hand in your `Date.java`.

Hint: You will find a multi-branched `if` statement useful for this problem. Also consider introducing an auxiliary method with the following signature:

```
// Examples: month("April") returns 4
//           month("July") returns 7
public static String month (String m);
```