CIT 594 – Homework 1

Due – Feb 3, 2016 at 12.00pm

Part 1 – Theory (20 points)

Please do the following problems:

1. Consider an array A of size n >= 2. The array contains numbers from 1 to n-1 (both inclusive) with exactly one number repeated. Describe an O(n) algorithm for finding the integer in A that is repeated. (5 points)
2. What is the order of growth for the following (using the Big-Oh notation)? (5 points each)

```
a. for(int i=1; i<n; i++) {
       for(int j=1; j<5; j++) {
           sum = sum + 1;
       }
   }
b. for(int i=1; i<n; i++) {
       for(int j=1; j<i*i; j++) {
           if(j%i == 0) {
               for(int k=0; k<j; k++) {
                   sum = sum + 1;
               }
           }
       }
   }
```

3. Consider the following two functions and the claim that f(x) is O(g(x)):
   f(x) = 3x^4 + 5x^3 + 17x^2 + 13x + 5; g(x) = x^4
   Prove whether the claim is true or false using the definition of Big-Oh. (5 points)

Part 2 – Programming (80 points)

Battleship: Battleship is a 2-player board game. You can find out background and rules about it here http://en.wikipedia.org/wiki/Battleship_%28game%29.

For the purpose of this assignment, you will implement a simpler version of the game in Java. The game rules are described below:

1. The game uses a 10x10 board.
2. At the start of the game, each player will (strategically) place 5 ships on the board. There are no constraints on the placement of the ships, except, of course, that they can't overlap each other

or go out of the board. The ships can only be placed horizontally or vertically (no diagonals allowed). The ships have the following lengths
    a. Aircraft Carrier – 5
    b. Battleship – 4
    c. Submarine – 3
    d. Cruiser – 3
    e. Destroyer – 2
3. Once the ships have been placed on the board, the game begins.
4. The players take turns starts calling out shots on the board. If a ship occupies the spot, it's a "Hit". Otherwise, it's a "Miss".
5. When a ship has been hit at all the board positions, it is "Sunk".
6. The game ends when all the ships (of one player) have been sunk.
7. The winner is the player who still has ships left on the board.

To make things easier (and for the tournament – see EC below), interfaces have been provided. Your code needs to implement these interfaces. They are described below:

1. `Game` – "The `Game` interface - this will control the Battleship game. It will keep track of 2 versions of the "board" - one for each player. It will let players take turns. It will announce hits, misses, and ships sunk (by calling the appropriate methods in the Player interface/class)."
2. `Player` – "The `Player` interface - Each player will get to choose where to place the 5 ships and how to take turns shooting at enemy ships."
3. `Location` – "The `Location` interface to specify how x and y coordinates are represented. This can be used to represent the location of a ship or a shot. If the location is a shot, the `isShipHorizontal`() method can return an arbitrary value."

Requirements: Everyone needs to implement a class that implements the `Game` and `Location` interface. Everyone needs to implement at least 2 classes that implement the `Player` interface – there should be a `HumanPlayer` (that will ask a user to enter coordinates for shooting, show the status of the board, etc. on the command line) and a `ComputerPlayer` (that will use a "strategy" for playing).

Part 2 – Extra Credit (10 points)

We will have an in-class tournament (to be run by the TAs after the homework submission date).

+1 – If you decide to participate in the tournament, you will get 1 EC point (assuming your code adheres to the specified interfaces and doesn't crash on running). Please indicate yes/no in the readme.txt file.
+4 – If you finish in the top 50%, you will get an extra +4 points.
+2 – If you finish in the top 25%, you will get an extra +2 points.
+3 – If you finish in the top 5%, you will get an extra +3 points.

For the EC part, you cannot have any help from the TAs/instructor.

Grading Criteria

5% for compilation – If your code compiles, you get full credit. If not, you get a 0.
75% for functionality – Does the code work as required? Does it crash while running? Are there bugs? …
10% for design – Is your code well designed? Does it handle errors well? …
10% for style – Do you have good comments in the code? Are your variables named appropriately? ...

Programming – General Comments

Here are some guidelines with respect to programming style.

Please use Javadoc-style comments.

For things like naming conventions, please see Appendix I (Page A-79) of the Horstmann book. You can also install the Checkstyle plugin (http://eclipse-cs.sourceforge.net/) in Eclipse, which will automatically warn you about style violations.

Submission Instructions

We recommend submitting the theory part electronically also. However, you can turn in a physical copy at the start of class, if you prefer. Please **do not** print out the Java source.

In addition to the theory writeup, you should also submit a text file titled readme.txt. That is, write in plain English, instructions for using your software, explanations for how and why you chose to design your code the way you did. The readme.txt file is also an opportunity for you to get partial credit when certain requirements of the assignment are not met. Think of the readme as a combination of instructions for the user and a chance for you to get partial credit.

Please create a folder called YOUR_PENNKEY. Places all your files inside this – theory writeup, the Java files, the readme.txt file. Zip up this folder. It will thus be called YOUR_PENNKEY.zip. So, e.g., my homework submission would be swapneel.zip. Please submit this zip file via canvas.