Name: _____ class & section _____

You must:

- **follow all style guidelines in Appendix I** and **F:** provide full javadoc style comments for each method and class (search the internet for more details). Do **NOT** create a java package or .jar
- spell names **exactly** as they are in these instructions so my tester will work with your class
- provide **both electronic and physical deliverables** (See below).

The Pickup Game explained:

This game has a number of variants. The following variant has an interesting winning strategy. Two players alternately take stones from a pile. In each move, a player chooses how many stones to take. The player must take **at least o**ne but **no more than half of the stones**. Then the other player takes a turn. The player who takes the last stone loses.

Write a program in which the computer plays against a human opponent. Generate a random integer between 8 and 55 to denote the initial size of the pile. Generate a random integer between 0 and 1 to decide whether the computer or the human takes the first turn.

Generate a random integer between 0 and 2 to decide whether the computer plays smart or stupid or alternates between smart and stupid.

In stupid mode, the computer simply takes a random legal value (between 1 and n/2) from the pile whenever it has a turn. In smart mode the computer takes off enough stones to make the size of the pile a power of two minus 1—that is, 3, 7, 15, 31. That is always a legal move, except when the size of the pile is currently one less than a power of two. In that case, the computer makes a random legal move. Note: the computer cannot lose in smart mode when it has the first move, unless the pile size starts as 7, 15 or 31. A human player who has the first turn and knows the winning strategy can win against the computer.

PickupGameTester (you must show expected result for every case/input):

This class' main method must instantiate an object of the PickupGame class and an object of the Scanner class. The version you submit must do at least the following:

1. instantiate an object of the Scanner class and use it to get **all** input (use may use a file or redirection if you research how to do so properly)
2. instantiate an object of the PickupGame class name **goodGame** (this PickupGame will demonstrate a wide range of usual, "good input")
3. Print all possible output/statistics for data collected so far (with expected values) for **goodGame**
4. Use tests to show that your code properly manages a wider variety of **valid but extreme or unacceptable input** (for example, with zero or only one game)
    a. instantiate an object of the PickupGame class name **extremeGame**
    b. Print all possible output/statistics for data collected so far (with expected values) for **extremeGame**
5. prove that your code properly manages a wide variety of bad input data, invalid type or no data
    a. instantiate an object of the PickupGame class name **badGame** (you may include some good input if needed to get output)
    b. Print all possible output/statistics for data collected so far (with expected values) for **badGame**. When/if statistics/output is needed, calculate and produce output showing the statistics for **all instance variables and all data** read in for the object at the current point in the program.
6. Since the input data might cause an exception, (non-numeric input), it should be encased in a try block, and have an accompanying catch block for the possible IllegalArgumentException (you can use and if statement). **See section 11.4**. Print out a message noting that the data is out of range / being ignored, and let the program continue if possible.

PickupGame class: must have the following instance variables with these names:

1. **initialStones**
2. **currentStones**
3. **playMode** :stupid, smart, alternating; use the random generator to select this
4. counters for each type/mode of gamed played: smart or stupid or alternates, such as **stupidCount**
5. computerWins for each type/mode of gamed, such as **stupidWin**
6. humanWins for each type/mode of gamed by the computer , such as **humanStupidWin**
7. ~~tiedGames for each type/mode of gamed by the computer, such as **stupidTieCount**~~
7. history - a String holding all valid moves for the **current** game entered by both players so far (See printReceipt method of E3.6 CashRegister class)

The class must have at least the following methods with these names:

1. a default constructor **PickupGame()** that **initializes all instance variables properly**
2. a constructor with the number of stones to start with as a parameter
3. public String toString (see @see String.format() p.A-27, format specifiers, p.524-5 and p.148-150) must return a multi-line String containing :
   3.1. **all** instance variables with suitable, brief headings
   3.2. the percentage of games won by human and by computer for each type/mode of gamed
   3.3. the percentage won by human for each type/mode of gamed (to 2 decimal places)

note: if a count is 0, do not divide by zero

You must submit all physical and electronic Deliverables. See Canvas for due date.

Physical: All Pages must be stapled in order (print on both sides if possible to save paper).

1. All pages of these instructions (printed from the web), with your **name clearly printed** on it, as a cover sheet.
2. Printed Source Code with Comments (including all javadoc style comments, **no line wrapping**)
3. Sample Input and Output screen captures (printed)
4. a simple **test plan** including explanations of any discrepancies and reasons for each test.
   4.1. Show actual input and include all of the outputs (expected & actual) in the test plan
   4.2. **ALL** values output as well as **ALL** expected output.
   4.3. Remember to test an <Enter> for expected inputs, a non-numeric character and just **one** numeric input.
   4.4. test multiple **sets** of data in one run.

Electronic:

5. Copies **of #3 and #4-4.4 above** as an rtf, All **.html (javadocs)**, and **.java** files **all** together as **one .zip** file. Do not use **rar** or any other archive format
6. Name the file: "<YourName>_p<#>.zip".
7. Submit this single **zip** file on **Canvas**

Extra Credit **(for a maximum of 5% of your initial earned score):**

Demonstrate and note in your class javadoc class heading that you enhanced the required contents & fully tested the following: at the start and after each move show an understandable "text graphic" view of the stones left. (You might use asterisk or O characters, showing the human's stones on e side, the computer's stones on the other and the stones still in the pile in the center.) for example, part way through a game you might see this:

Human stones: 6	in pile stones: 4	computer stones: 3
** ** **	** **	** *