CS 1331 Homework 9

Due Thursday, March 28th , 2013 8:00pm

Introduction

This assignment is all about exceptions and recursion. It is split up into three discrete problems you must solve: one deals with exceptions and two deal with recursion.

Unlike the recent assignments, you will not have one cohesive program at the end. Instead, you will have two separate programs. One that acts as a simple calculator, and one that contains the recursive solutions to two problems (determining the number of handshakes necessary if everyone in a room is to shake hands with one another, and determining the hailstone number sequence for a given number).

For the recursive problems, you will only need to write a single method for each. You should place those methods in a java file named Recursion.java. To test your methods, you should include a main method in which you call each method.

9.1 A Simple Calculator

This program is to act as a simple calculator. The calculator should keep track of a single double called `result` which starts at 0.0. Each cycle of prompting the user should allow the user to modify the result through addition, subtraction, multiplication, or division. For every cycle, the user should be prompted for the operation to perform and the second number to complete the operation.

Our program will only handle the following operators: *, /, +, -

If a user tries to perform an operation with something other than one of those operators your program should throw (and then handle) an `UnknownOperatorException`. This exception is one that you will have to create yourself.

When thinking about how to solve this problem, make sure you think in terms of object oriented design. As in all of the prompting based assignments we have had, there will be a reasonable amount of code in the main method, which should be located in `SimpleCalculatorDriver.java`, but you should also create an actual `SimpleCalculator` class that keeps track of the result and handles performing the operations based on the operator and number passed to it. The method that handles your operations should most likely be the one that throws the `UnknownOperatorException`, which will then be caught in the main method.

In addition to handling `UnknownOperatorException`s, your program also needs to handle `InputMismatchException`s (these are generated when your scanner expects and int, but receives something else) and `ArithmeticException`s (on divide by 0).

Here is sample output of the Simple Calculator program:

```
result = 0.0
Enter an operation or 'q' to end the program
+
Enter a number
3
result = 3.0
Enter an operation or 'q' to end the program
/
Enter a number
0
You cannot divide by 0!
result = 3.0
Enter an operation or 'q' to end the program
*
Enter a number
100
result = 300.0
Enter an operation or 'q' to end the program
%
Enter a number
10
We do not support that operation!
result = 300.0
Enter an operation or 'q' to end the program
*
Enter a number
a
That was not a number! Try again.
10
result = 3000.0
Enter an operation or 'q' to end the program
q
```

9.2 Handshake Problem

To solve this problem, you will need to write a method called handshake that recursively determines the number of handshakes needed for all of the individuals in a room to shake everyone else's hand.

This method has the number of people in the room as a parameter and returns the number of handshakes needed. The method should be static since it does not require any knowledge or manipulation of any instance variables within the class.

As stated in the introduction, you should write this static method in Recursion.java. To test it, call the method from within the main method of Recursion.java and print out the results. Include several test cases in your main method. It must work for any integer values greater than or equal to zero. Having good test cases is part of the task!

9.3 Hailstone Problem

The last problem in this assignment is to write a recursive method called hailstone that prints out the hailstone number sequence for a given input number that is a positive integer. If you are unfamiliar with hailstone numbers, check out this link : http://en.wikipedia.org/wiki/Collatz_conjecture

Essentially, the hailstone sequence for a given number is a pattern that results when you perform certain operations on the number whether it is even or odd. You record the number at every step, and eventually the number becomes 1.

When the number(n) is even, you divide it by two (n/2). When the number is odd, you multiply it by 3, and then add 1 (3*n +1).

You will write another static method in Recursion.java to solve this problem.

Your method should be named hailstone. It should take in the number as a parameter, and return the modified number (n/2 or 3*n+1). Every time the method is called, it should print the number to the console before it does anything else with it.

Again, you should call this from the main method with various initial numbers to test to see if it performs as expected. How good your test cases are will be graded. Be sure your method works for integer values greater than or equal to 1.

Sample Output:

When hailstone(5) is called:

```
5
16
8
4
2
1
```

When hailstone(40) is called:

```
40
20
10
5
16
8
4
2
1
```

When hailstone(29) is called:

```
29
88
44
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
```

Turn-in Procedure

Turn in the following files to T-Square. When you are ready, make sure that you have actually **submitted** your files, and not just saved them as a draft.

- SimpleCalculator.java
- UnknownOperatorException.java
- SimpleCalculatorDriver.java
- Recursion.java

Note** Always submit .java files - never submit your .class files. Once you have submitted and received the email from T-Square, you should download your files into a fresh folder. Compile, run, and test those exact files that you turned in. It is pretty much foolproof if you use this technique. Anything less than this is a gamble. See "safe submission" info below. File issues, non-compiling files, non-source code files, etc are all problems and will cause a 0 for the HW. Also, make sure that your files are in on time; the real deadline is 8 pm. While you have until 2 am to get it turned in, we will not accept homework past 2 am for any reason. Don't wait until the last minute!

Verify the Success of Your HW Turn-in

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. Read that email. Look at those filenames. We do not grade .class files. We require source code .java files.
3. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files and the fact that you submitted.
4. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
5. Recompile and test those exact files.
6. This helps guard against a few things.
    a. It helps insure that you turn in the correct files.
    b. It helps you realize if you omit a file or files. (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    c. Reading the email and looking at the files you download helps prevent the turn-in of bytecode (.class) file for which you will receive no credit.
    d. Helps find last minute causes of files not compiling and/or running.