Recursion Assignments (pattern, hourglass, commas)
CS 162 - Spring 2018

Goal
- to write several recursive programs
- to write programs from scratch with a console interface
- to work alone on solving problems (no pairs allowed)
- pattern and hourglass are due by 11pm Monday, April 23. Each is worth 50 points.
- commas is due by 11pm Wednesday, April 25. It is worth 75 points.

You are not allowed to get help from or talk to anyone about this assignment.

Overview
In these three assignments, you will write Java programs that each contain a recursive method. Read the specifications for each carefully. Your messages need to match ours.

The first, pattern, prints a numeric pattern such as this for input 4: 

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

The second, hourglass, prints a symmetric graphic with text. See back. The third, commas, prints a long integer with commas in the correct places. The names of the classes containing main should start with upper case letters: Pattern, Hourglass, and Commas.

Classes (other than the needed class containing main), arrays, loops, stacks, queues, pointers, and linked lists are inappropriate here and should not be used in the recursive methods. The pattern and hourglass programs should not contain any loops. The commas program will have an input loop but no other loops. Do not use data structures beyond Strings in the hourglass program. Do not use any operations on Strings other than concatenation. In particular, do not use any substring or subscripting operations. Do not ask for its length.

Include comments at the top of your programs and on any method other than main.

You should NOT use any variables declared outside the methods of the class. All communication with the methods should be through parameters and return values.

Carefully note the restrictions on language feature use in the specifications for each program.

Pattern Specification
The program reads a single integer from the user. If that integer is less than 1, it prints the message “Input is less than 1” and stops. Don’t ask for a better value. Stop. Otherwise, it prints a pattern of integers as demonstrated in the table below. Print a newline after reading the input and before printing the answer. Print another newline after printing the answer. There is a space before each integer, a newline after each integer greater than 5, and the underlines and boldface are hints to help you see that pattern and are not printed by your program. Your program should work for any positive integer, but because it prints $2^n-1$ numbers (plus spaces) for input n, do not test it with anything bigger than 15.

<table>
<thead>
<tr>
<th>n</th>
<th>what is printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1 2 1</td>
</tr>
<tr>
<td>3</td>
<td>1 2 1 3 1 2 1</td>
</tr>
<tr>
<td>4</td>
<td>1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 6</td>
</tr>
</tbody>
</table>

This produces a numeric pattern symmetric about n where each half before and after n is also symmetric around n-1. The newline after printing any integer larger than 5 provides a nice line length and also makes the pattern of the bigger numbers along the right edge reading down. If you get the normal pattern, that will just happen. Don’t think about it. Output for an input of 7 is:

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 6 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 7 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 6 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 7

This program must not contain a loop.

Approaching the Pattern Problem
Obtain the input and check its validity in the main method. In main, put a newline both before and after your recursive method’s total output.

Write a public static method that has a void return type and takes a single integer as its one parameter, n.

In writing recursive methods, consider the base case. What is the base case here? It’s the simple case that doesn’t require another method call. For this problem, it’s what happens when n is 1. Write that case and test it.

Now what happens if n is larger than 1?

Test your program thoroughly. You can test it exhaustively up to 10, but it should work for larger numbers. Be careful on much larger numbers. When you have completed the program, submit it as the pattern assignment.
**Hourglass Specification**

The program reads a single integer from the user. If that integer is less than 1, it prints the message "Input is less than 1" and stops. The hourglass base case with input 1 displays two lines with one star on each line. Otherwise, it prints a pattern of asterisks or stars (*) as shown below for an input of 4. Note that there are four stars and four spaces on the first and last lines, with one space after each star.

```
* * * *
* * *
* *
* *
* * *
* * * *
```

The program must not contain a loop.

**Approaching the Hourglass Problem**

Obtain the input and check its validity in the main method. In main, put a newline both before and after your recursive method's total output.

Write a public static recursive method. There are many ways to write this method, but most of them involve having two parameters representing the number of stars and the number of spaces in front of the stars on the top and bottom lines. For the overall hourglass design, there are no spaces before the stars on the first and last lines.

It will be extremely helpful to write a supporting method called `printMany` that takes in an integer count and a String s. It prints count copies of s without finishing the line. I strongly suggest you write that method first and call it from main. You wouldn't normally write such a method recursively in Java, but this is recursion practice. Do even that recursively. No loops.

Test your program thoroughly. When you have completed the program, submit it as the `hourglass` assignment.

**Commas Specification**

Your program should contain a recursive method that takes a long value and returns a String of the value with commas separating clumps of three digits as shown below. Use `/` and `%` to break the number apart. The method should deal with negative numbers (by concatenating a String with the - character with the result of making a recursive call with the negation of the original number), with numbers that are less than 1000 (by concatenating the empty string with the number), and with numbers that are greater than or equal to 1000 (by recursively calling itself with the number divided by 1000 and then concatenating the last three digits).

The main method should repeatedly call this method with the long integers that are read from the input until there are no more long integers in the input. Use a while loop in the main method. Do not call the main method recursively. Do not prompt for more input in the loop. A prompt written once using println before the loop is acceptable. Each formatted integer should be printed on a separate line with no spaces. Note that to receive any credit, you must use a recursive method as outlined above.

You may not use NumberFormat or DecimalFormat or any other formatting class. This is a recursion assignment.

Note also that the program is limited to numbers that are between $-(2^{63} - 1)$ and $2^{63} - 1$. Inputs outside that range will not give correct results. Your program should not assume anything about this limit. That is, the program should work correctly with the types changed to a type that allows unlimited size numbers. Do not check the range.

**Approaching the Commas Problem**

- method names should describe what the method accomplishes for the user and not how the method is implemented. Therefore, for this assignment, any name involving forms of the word recursion is inappropriate.
- you should declare the input variable and the parameter to your recursive method to be of type `long`.
- use ifs and else-ifs (not DecimalFormat) to fill with zeros where needed; carefully consider the cases.
- after you have read and printed a number, you can forget about it. There is no limit to the number of values that may be read. There is no sentinel other than end-of-file (Control-D).
- if x is less than 0, -x is its absolute value and is positive. Do not multiply by -1.
- you absolutely must use a recursive method as described above to receive credit.
- the last two examples are the largest and smallest numbers that will work.
- the following lines are part of an unacceptable solution:
  ```java
  import java.text.DecimalFormat;
  private static DecimalFormat doItAll = new DecimalFormat("#,###");
  System.out.println(doItAll.format(num));
  You may not use NumberFormat or DecimalFormat to format the output.
  ```

When you have completed the program, submit it as the `commas` assignment.

**Example Input (Bold) and Output**

```
1024
1,024
42
42
0
0
-365
-365
-76543
-76,543
12345678
12,345,678
-1,234,567,890,123,456,789
-12,345,678,901,234,567,890
9,223,372,036,854,775,807
9,223,372,036,854,775,807
9,223,372,036,854,775,807
9,223,372,036,854,775,807
```