CPS 112: Computational Thinking with Algorithms and Data Structures
Homework 5 (20 points)

Handed out: October 25, 2017 (Wednesday)
Due: 11:59 pm November 1, 2017 (Wednesday)
Late submissions accepted with penalty until 11:59 pm November 3, 2017 (Friday)

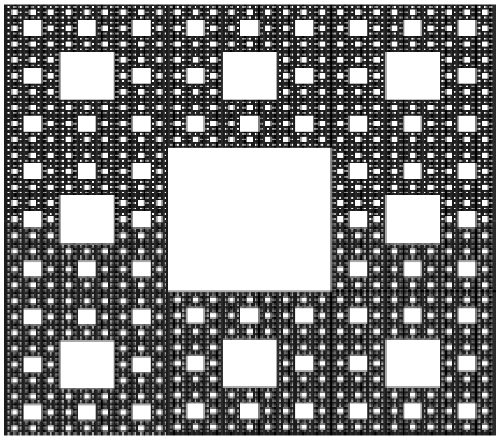
If you run into trouble or have questions, arrange to meet with me or a tutor!

Before you submit:

- **Please review the homework guidelines available on Canvas**
- Read the assignment very carefully to ensure that you have followed all instructions and satisfied all requirements.
- Create a zip file named like this: *yourusernameHWX.zip* (with your username and replacing the *X* with the assignment number) that contains a directory named *yourusernameHWX*, which contains all of your .java files, and your cover sheet.
- Make sure to compile and thoroughly test all of your programs before you submit your code. **Any file that does not compile will receive a 0!**
- Ensure that your code conforms to the style expectations set out in the homework guidelines.
    - Make sure your variable names are descriptive and follow standard capitalization conventions.
    - Put comments wherever necessary. Comments at the head of each file should include a description of the contents of the file **(no identifying information please!)**. Comments at the beginning of methods describe what the method does, what the parameters are, and what the return value is. Use comments elsewhere to help your reader follow the logical flow of your code.
    - *Program readability and elegance are as important as correctness.* After you've written your function, read and re-read it to eliminate any redundant/unused lines of code, and to make sure variable and function names are intuitive and relevant.

1. (3 pts) In *Palindrome.java* you have been given a recursive method to determine if a `String` is a palindrome or not. You should notice that it is rather inefficient because it creates substrings on each pass. Re-write the method so that it still recursively determines if the `String` is a palindrome or not, but without using substrings (**No cheating... no converting to an array or anything else either**).

2. (7 pts) In this problem you will write a function that generates a Sierpinski carpet. A Sierpinski carpet is like a Sierpinski triangle, except its structure is based on squares rather than triangles:

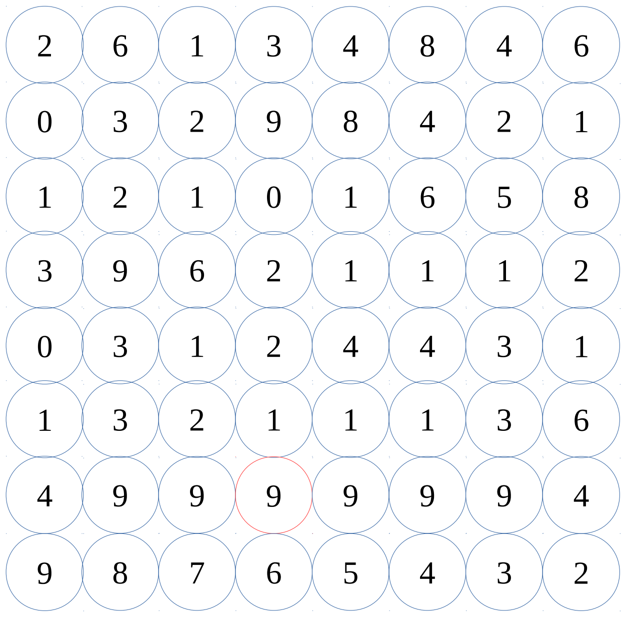


   In *Sierpinski.java* fill in the *recursive* method `drawSquare` to generate a Sierpinski carpet. Your method should take four parameters: a `Point` object representing the upper-left corner of the square to be drawn, a `Point` object representing the lower-right corner of the square to be drawn, an integer representing the "level" of the carpet (level 0 is just a plain square), and a `Graphics` object on which to draw the carpet. Use the Sierpinski triangle code from lecture as a model. Test your code by calling *java Sierpinksi 4* to draw a carpet similar to the one above.

   I strongly suggest you work out by hand one or two levels so you know where the points go. The drawing region is 243 x 243 (roughly square) where (0, 0) is the upper-left hand corner. These dimensions aren't quite exact so the numbers may not exactly match if you print them out, but they should be good enough for you to work out the code correctly.

3. (10 pts) Everybody loves bubbles, and I've got some special bubbles that contain money. Consider group of bubbles arranged on the next page. Given a specific starting bubble, (like the one marked in red) the goal is to pop bubbles in order to obtain the most amount of money based on the following rules.
   1. You pop the start bubble
   2. You may then continue to pop bubbles as long as it is a bubble that is northwest (up one row and one column to the left) or northeast (up one row and one column to the right) from the bubble you just popped. That is, each time you pop a bubble, you should be one row closer to the top. Also note that you DO NOT always have to go the same direction.
   3. Once you pop a bubble in the top row, you stop

   Given the example below, the optimal solution is 37, by moving NE, NE, NE, NW, NW, NE ending up on position (0, 5). The sum of values along this path is 9+1+4+1+6+8+8.



a) In the provided *Bubbles.java*, write a recursive method to find the max value. In that file, I have provided you with the bubble setup from above. You do not need to give the path, only the maximum amount of money that can be collected. **Hint:** One of the base cases is if a movement to one of the diagonal positions is out of bounds, return 0. You will need to figure out the other base case as well as the recursive case.

b) If you were to trace through the recursive calls by hand you would see that the same bubble gets checked several times especially the closer to the top it is. Memoization can help to eliminate these redundant calls. Convert your current solution to a solution that uses memoization. **Hint 1:** You will need a double array and not a single array.
Hint 2: The base case regarding diagonal moves off the grid should remain the same as in a) above, but the rest of your recursive solution should use memoization.

You only need to submit your memoized solution.