Stacks and Queues

(1) Read through the following code. Draw the states of the variables stack and queue when indicated.

```java
Stack<String> stack = new CarlStack<String>();
Queue<String> queue = new ArrayDeque<String>();
queue.add("tofu");
queue.add("broccoli");
queue.add("rice");
// (a) Draw the states of stack and queue
System.out.println(queue.element());
while (!queue.isEmpty()) {
    stack.push(queue.remove());
}
// (b) Draw the states of stack and queue
for (int i = 0; i < 2; i++) {
    queue.add(stack.peek());
}
// (c) Draw the states of stack and queue
```

(2) We briefly talked about post-fix notation. Brainstorm with your partner how to evaluate a post-fix notation expression using a stack; that is, find the final value that the expression represents. Write your steps down in pseudocode (step-by-step English). Assume your algorithm will take a list as input. [Hint: Read the next question.]

create an empty stack
for each item in list:
    if item is binary operator op:
        second = pop off stack
        first = pop off stack
        answer = evaluate "first op second"
        push answer onto stack
    else:
        push item onto stack
finalAnswer = pop off stack
return finalAnswer

(3) You receive the following list as input to your algorithm: [3, 4, 2, /, 8, +, *]. Walk through the steps of your algorithm with this input, and draw the state of the stack after each pop or push.

```
(over)
(4) **Palindromes** are words or phrases that read the same forward and backward, ignoring spaces, punctuations, or capitalisation (e.g., “racecar” or “Sit on a potato pan, Otis!”). How would you use a stack and a queue to check whether a word or phrase is a palindrome? Write down an algorithm (in pseudocode) to check if something is a palindrome using these two data structures.

create 1 stack and 1 queue

go through the string in order, one character at a time
  if it is not a letter, ignore and go on to next letter
  convert letter to lower case
  push it onto the stack and add it to the queue
afterwards, process stack and queue
while the stack is not empty
  pop the top item from the stack and remove the first item from the queue
  if they differ, return false
if the stack is exhausted, return true

Code:

```java
public static boolean isPalindrome(String s)
{
    Stack<Character> stack = new CarlStack<Character>();
    Queue<Character> queue = new ArrayDeque<Character>();

    for (int i = 0; i < s.length(); i++)
    {
        char character = s.charAt(i);

        // ignore non-letter characters
        if (Character.isLetter(character))
        {
            // ignore upper/lower case differences
            character = Character.toLowerCase(character);
            stack.push(character);
            queue.add(character);
        }
    }

    while (!stack.isEmpty())
    {
        if (stack.pop() != queue.remove())
        {
            return false;
        }
    }
    return true;
}
```

Extra time? Think about how to evaluate *infix* expressions, convert between infix and postfix, implement a queue using two stacks, implement a stack using two (or one!?) queues; or write Java code to implement your pseudocode algorithms.