CSC 212 Programming Assignment # 2
Implementing and Using Queues and Stacks
Due date: 02/27/2019

Guidelines:	This is an **individual** assignment. The assignment must be submitted to **Web-CAT**

1. The first part consists in the implementation of the ADT Queue and Stack.

   (a) Given the following specification of the ADT Queue, implement this data structure using **linked representation**. You should write the class `LinkedQueue` that implements the interface `Queue`.

   - enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
   - serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
   - length (int l): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: l.
   - full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.

   New methods:

   - multiEnqueue(Type els[], int k, int l): **requires**: None. **input**: Type els[], int k. **results**: The first k elements of the array els are added to the queue at its tail one at a time until the queue is full or all k elements are added. The output l is set to the number of elements that have been added. **output**: l.

   (b) Given the following specification of the ADT Stack, implement this data structure using **array representation**. You should write the class `ArrayStack` that implements the interface `Stack`.

   - push(Type e): **requires**: Stack S is not full. **input**: Type e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
   - pop(Type e): **requires**: Stack S is not empty. **input**: **results**: the most recently arrived element in S is removed and its value assigned to e. **output**: Type e.
   - empty(boolean flag): **requires**: none. **input**: none. **results**: If Stack S is empty then flag is true, otherwise false. **output**: flag.
   - full(boolean flag): **requires**: none. **input**: none. **results**: If S is full then Full is true, otherwise Full is false. **output**: flag.

New methods:

- multiPush(Type els[], int k): **requires**: None. **input**: Type els[], int k. **results**: The k elements of the array els are pushed to the stack one at a time until the stack is full or all k elements are added. **output**: none.
- reverse(): **requires**: none. **input**: none. **results**: The entire stack S is reversed **output**: none.
- copy(Stack: ns): **requires**: none. **input**: none. **results**: Create ns, a copy of S. S does not change. **output**: ns.

2. Write a class called `ExpUtils`. In this class, you should implement the following static methods:

   - **public static** `Queue<Stack<Character>> readExps(String fileName)`: A method that reads expressions from a text file `fileName` separated by new line and builds a queue of stacks, where each expression is represented by one stack of characters (assume that all numbers are single digit and that there are no spaces in the expression). The first expression in the file must be at the beginning of the queue. The beginning of the expression must be at the top of the stack. The method must catch any exceptions and return null if any.

     Sample input:

     ```
     (5+6)/8
     (2/5)*9
     (9*3)/(5+2)
     (5*6)/4
     ```

   - **public static** `Queue<Stack<Character>> match(Queue<Stack<Character>> q, String pattern )`: searches the queue `q` for all expressions that match `pattern` and return a copy of them stored in a queue. The order of the output must be the same as that of `q`, and `q` must not change. In `pattern`, the special character '.' matches any (single) character.

     Example 1. *Given the above set of expressions, the pattern "*`(5.../.`*" matches two expressions:*

     ```
     (5+6)/8
     (5*6)/4
     ```

3. In a number pyramid (Figure 1), each number is the sum of the two numbers on which it stands.

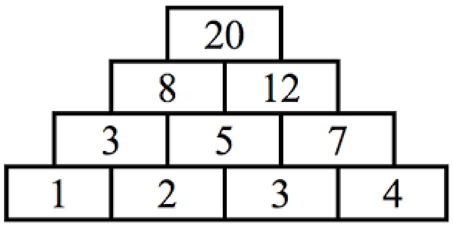

Figure 1: An example of a number pyramid.

Write a class called `NumPyramid`. In this class, you should implement the following static methods:

- **public static** `Stack<Queue<Integer>> build(Queue<Integer> q)`: takes as input a queue of numbers representing the leftmost bricks of the pyramid (for Figure 1, $a = \{20, 8, 3, 1\}$) and returns the whole pyramid represented as a stack of queue. Each queue represents a row of the pyramid. The top of the pyramid must be a t the top of the stack. The input queue must not change.

1 Deliverable and rules

You must deliver:

1. Source code submission to Web-CAT. You have to upload the following class in a zipped file:

   - `LinkedQueue.java` (include class `Node` here).
   - `ArrayStack.java`
   - `ExpUtils.java`
   - `NumPyramid.java`

   Notice that you should **not upload** the interfaces.

The submission **deadline** is: **02/27/2019**.

You have to read and follow the following rules:

1. The specification given in the assignment (**class and interface names, and method signatures**) must not be modified. Any change to the specification results in compilation errors and consequently the mark zero.
2. All data structures used in this assignment **must be implemented** by the student. The use of Java collections or any other data structures library is strictly forbidden.
3. This assignment is an individual assignment. Sharing code with other students will result in harsh penalties.
4. Posting the code of the assignment or a link to it on public servers, social platforms or any communication media including but not limited to Facebook, Twitter or WhatsApp will result in disciplinary measures against any involved parties.
5. The submitted software will be evaluated automatically using Web-Cat.
6. All submitted code will be automatically checked for similarity, and if plagiarism is confirmed penalties will apply.
7. You may be selected for discussing your code with an examiner at the discretion of the teaching team. If the examiner concludes plagiarism has taken place, penalties will apply.