CSC 212 Programming Assignment # 2
Implementing and Using Queue and Stack
Due date: 31/10/2018

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

The goal of this assignment is to implement and use both the Queue and Stack ADTs. The assignment is divided into two parts. In the first part, you will implement Queue and Stack with an augmented set of operations. In the second part, you will write methods that use this implementation:

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 **Array representation**. You should write the class `ArrayQueue` that implements the interface `Queue`.

   ## Specification of ADT 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.
   - multiServe(Type els[], int k, int l): **requires**: None. **input**: Type els[], int k. **results**: The first k elements of the queue are served and stored in els starting at position 0 until the queue is empty or all k elements are served. The output l is set to the number of elements that have been served. **output**: l.

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

Specification of ADT 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, int l): **requires**: None. **input**: Type els[], int k. **results**: The first 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. The output l is set to the number of elements that have been pushed. **output**: l.
- multiPop(Type els[], int k, int l): **requires**: None. **input**: Type els[], int k. **results**: The top k elements of the stack are popped and stored in els starting at position 0 until the stack is empty or all k elements are popped. The output l is set to the number of elements that have been popped. **output**: l.

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

   (a) **public static** `Stack<Queue<Character>> readSentences(String fileName)`: A method that reads sentences from a text file `fileName` separated by new line and builds a stack of queues, where each sentence is represented by one Queue of characters. The first sentence in the file must be at the top of the stack. The method must catch any exceptions and return null if any.

   Sample input:

```
ABCDEFGHIJKLMN
I love CSC212
The fat cat sat on the mat
curiosity killed the cat
```

   (b) **public static void** `encrypt(Queue<Character> q, int k)`: A method that reverses each block of `k` characters in the sentence represented by `q`. Assume $k \geq 1$.

   Example 1. *Here are two examples:*

   - *If* `q = ABCDEFGHIJKLMN`*, then by calling* `encrypt(q, 4)`*, we obtain:* `DCBAHGFELKJINM`.
   - *If* `q = I love CSC212`*, then by calling* `encrypt(q, 4)`*, we obtain:* `ol IC ev12CS2`.

   Hint: use the new methods added to Queue and Stack.

   (c) **public static void** `decrypt(Queue<Character> q, int k)`: A method that decrypts (removes the encryption) the sentence represented by `q`. Assume $k \geq 1$.

   Example 2. *Here are two examples:*

   - *If* `q = DCBAHGFELKJINM`*, then by calling* `decrypt(q, 4)`*, we obtain:* `ABCDEFGHIJKLMN`.
   - *If* `q = ol IC ev12CS2`*, then by calling* `decrypt(q, 4)`*, we obtain:* `I love CSC212`.

(d) **public static void** `encrypt(Stack<Queue<Character>> st, int k)`: A method that encrypts all sentences in the stack `st`. The order of the sentences within the stack must not change. Assume $k \geq 1$.

(e) **public static void** `decrypt(Stack<Queue<Character>> st, int k)`: A method that decrypts all sentences in the stack `st`. The order of the sentences within the stack must not change. Assume $k \geq 1$.

1 Deliverable and rules

You must deliver:

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

   - `ArrayQueue.java`
   - `LinkedStack.java`
   - `SimpleEncrypt.java`

   Notice that you should **not upload** the interfaces `Queue` and `Stack`.

The submission **deadline** is: **31/10/2018**.

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.