Topic: Stack & Queue

Part A: ***Basic Concept***

1. Complete the following table that shows a series of stack operations and their effects on an initially empty stack S of integers.

Method	Return Value	Stack Contents
push(6)	-	(6)
push(4)	-	(4, 6)
size( )	2	(4,6)
peek( )	6	(4,6)
pop( )	6	(4)
isEmpty( )	false	(4)
pop( )	4	( )
isEmpty( )		
push(5)		
push(2)		
peek( )		
push(7)		
size( )		
pop( )		

2. Complete the following table that shows a series of queue operations and their effects on an initially empty queue Q of integers.

Method	Return Value	First←Q←Last
enqueue(6)	-	(6)
enqueue(4)	-	(6, 4)
size( )	2	(6, 4)
first( )	6	(6, 4)
dequeue( )	6	(4)
isEmpty( )	false	(4)
dequeue( )	4	( )
isEmpty( )		
enqueue(5)		
enqueue(2)		
first( )		
enqueue(7)		
size( )		
dequeue( )		

3. Write Java statement that creates:
   a) an object variable `intStack` that represent a stack of integers.

   b) an object variable `strStack` that represent a stack of string.

   c) an object variable `intQueue` that represent a queue of integers.

   d) an object variable `strQueue` that represent a queue of string.

4. Consider the following code fragment. Draw a diagram to represent the following stack operations, step-by-step as the program executes. Show the output, if any.

a)

```
Stack<String> s = new Stack<String>();
s.push("happy");
s.push("sad");
String st = s.peek();
s.push("numb");
s.push(st+"dle");
s.pop();
st = s.pop();
s.push(st);
```

b)

```
Stack <Integer> stack1 = new Stack<Integer>();
System.out.println(stack1.empty());
for (int i = 0; i < 5; i++)
      stack1.push(new Integer(i*10));
      System.out.println(stack1.peek());

Stack <Integer> stack2 = new Stack<Integer>();
while (!stack1.empty())
      stack2.push(stack1.pop());
      System.out.println(stack2.peek());
```

c)

```
Stack<Integer> s = new Stack<Integer>();
int num1, num2;

s.push(12);
s.push(5);
num1 = s.peek() + 3;
s.push(num1+5);
num2 = s.peek() ;
s.push(num1 + num2);
num2 = s.peek() ;
s.pop();
s.push(15);
num1 = s.peek();
s.pop();

while (!s.isEmpty()) {
   System.out.println (s.peek());
   s.pop();
}
```

5. Consider the following code fragment. Draw a diagram to represent the following queue operations, step-by-step as the program executes

```
Queue<Integer> q = new Queue<Integer>();
q.enqueue(5);
q.enqueue(7);
q.enqueue(13);
q.dequeue();
Integer t = q.peek();
q.enqueue(12+t);
q.dequeue();
q.enqueue(q.dequeue());
```

6. Consider the following code fragment. Draw a diagram to represent the following stack and queue operations, step-by-step as the program executes.

```
Stack <Integer> s = new Stack<Integer>();
for (int i = 0; i < 4; i++)
   s.push(new Integer(i*2+3));
Queue <Integer> q = new Queue<Integer>();
while (!s.empty())
   q.enqueue(s.pop());
while (!q.empty())
   s.push(q.dequeue());
```

7. Using the given Queue class ,
   i. Complete *MyQueue* class (main method) of the following code.

```
public class MyQueue{
   public static void main(String[] args) {
      //create a queue q of string
      _____
      //Add element "Azura" to the queue and display the queue
       q.enqueue("Azura");
       System.out.println(q);
      //Add element "Masura" to the queue
      _____
      //Display the elements of the queue
      _____
      //Add element "Rohizah" and "Faridatul" to the queue
      _____
      _____
      //Display the elements of the queue
      _____
      //Remove and display the queue
      _____
      //Remove and display the queue
      _____
      //Display the elements of the queue
      _____
      //Display the size of the queue
      _____
   }
}
```

   ii. What is the expected output of the above code?

Part B: *Stack & Queue Applications & Problem Solving*

8. Trace the algorithm given in lecture slide to determine the value of the following postfix expression.

   a. 15 4 3 - * 5 + =

   b. 2 3 + 4 6 3 / - * 5 + =

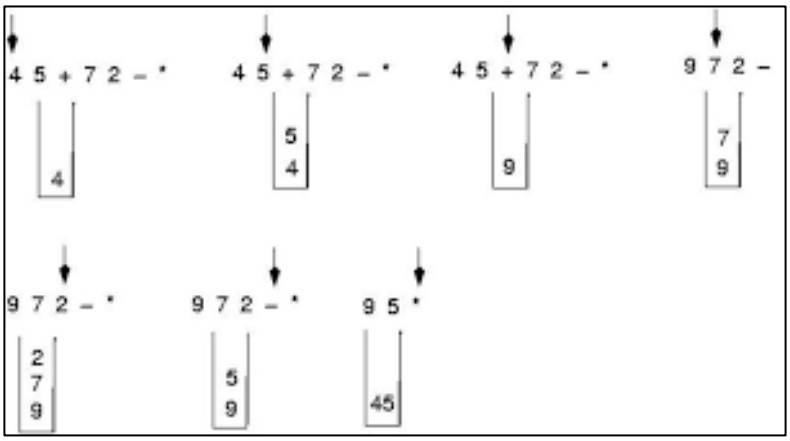
   c. 12 25 5 1 / / * 8 7 + - =

   d. 3 5 6 * + 13 - 18 2 / + =

   e. 10 2 * 5 + 5 / 8 + =

   Show your work as in the following figure or a simplified version based on your understanding.

Example:



9. Trace the algorithm given in lecture slide to convert the following infix expression in their postfix forms.

   a. D – B + C

   b. A * B + C * D

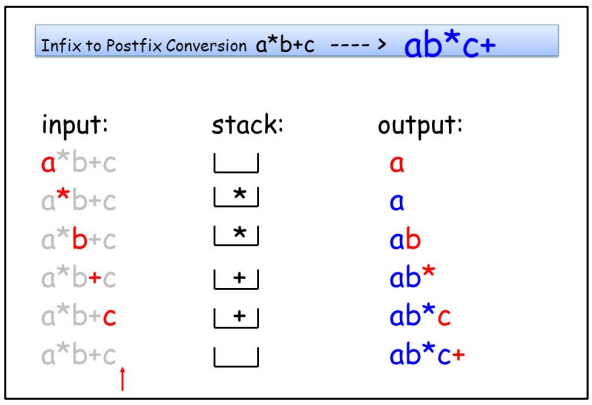
   c. (A + B) * C – D * F + C

   d. (A – 2 * (B + C) – D * E) * F

   e. A + B * (C-D) / (P-R)

   Show your work as in the following figure or a simplified version based on your understanding.

Example:



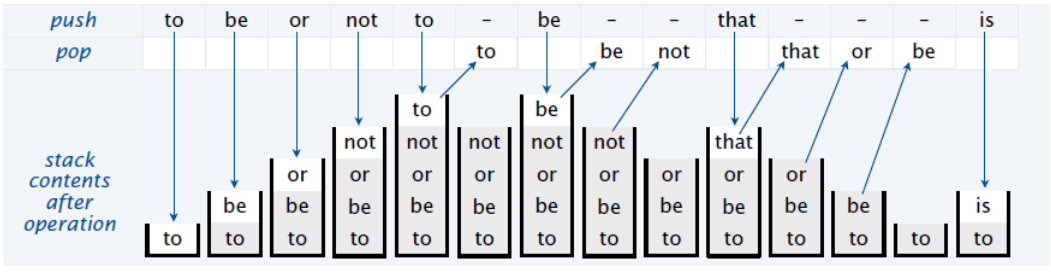
10. Write Java code segment that reads a sequence of integers that ends with 0. If the integer is odd, add to oddQueue, otherwise add to evenQueue. Display your output in the following format:
    oddQueue <size of queue>: <element lists>
    evenQueue <size of queue>: < element lists>

    Sample Input /Output:

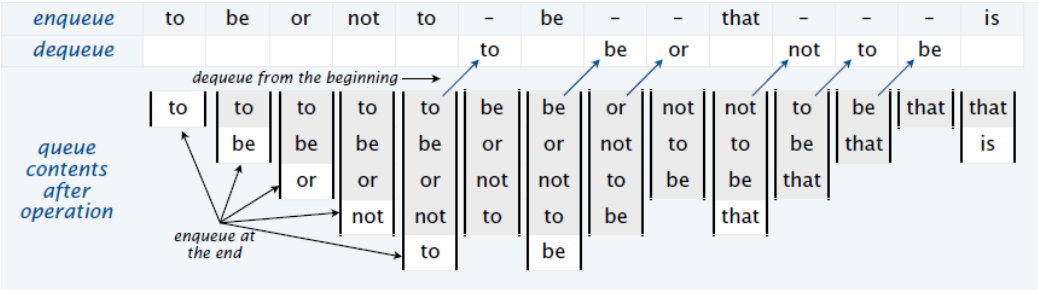
Input:	Output
34 1 8 5 22 0	oddQueue 2: 1 5 evenQueue 3: 34 8 22
8 16 9 4 27 -5 0	oddQueue 3: 9 27 -5 evenQueue 3: 8 16 4

11. Write Java code segment that reads a line of words including '-'. Add to the appropriate data structure when you see a word, and remove and print when you see a '-'. At the end of input, print the number of items left in the structure. Diagram below illustrate an example of how your program should work.

    a) stack



b) queue



12. Palindrome is a sequence of characters, which reads the same backward or forward (eg. *katak*). Implement (write Java code) the following Palindrome algorithm.

```
create 1 stack and 1 queue
go through the string in order, one character at a time
        push it onto the stack and add it to the 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
```

13. Implement the balance symbol checker algorithm given in lecture slide.