1. Describe and justify how you plan to implement the stack interface (e.g. nodes, arrays, save it all to file?). Use details from the problem to justify your answer.
   The stack Interface required methods are given on 268 wiki page. They are:
   - virtual ~StackInterface() {};

   - virtual bool isEmpty() const = 0;

     isEmpty is to check whether the stack is empty.

   - virtual void push(const T value) throw(PreconditionViolationException) = 0;

     Push is to add new entry to the top of stack. In LinkedList, we use addFront as the same method as push do. It also throw a warning when the field is full and cannot add any more values.

   - virtual void pop() throw(PreconditionViolationException) = 0;

     Pop is to remove the top stack. In LinkedList, we used removeFront as the same method as pop do. It also throw a warning when the stack is empty and nothing can be removed.

   - virtual T peek() const throw(PreconditionViolationException) = 0;

     Peek is to return value at the top of the stack. In LinkedList, we used getEntry as the same method as peek do, but it only get the first entry's value. It also throw a warning when the stack is empty.

2. Describe and justify how you plan to implement the queue interface (e.g. nodes, arrays, save it all to file?). Use details from the problem to justify your answer.

The Queue Interface required methods are given on 268 wiki page. They are:

- virtual ~QueueInterface() {}
- virtual bool isEmpty() const = 0;
  isEmpty is to check whether the stack is empty.

- virtual void enqueue(const T value) throw(PreconditionViolationException) = 0;
  Enqueue is to add new entry to the back of queue. In LinkedList, we used addBack to do the same thing. It will throw a warning when field is fulled and nothing can be added to queue.

- virtual void dequeue() throw(PreconditionViolationException) = 0;
  Dequeue is to remove the the first one in queue. In LinkedList, we used remove front to do the same thing. It will throw a warning when the queue is empty.

- virtual T peekFront() const throw(PreconditionViolationException) = 0;
  PeekFront is to returns the value at the front of the queue. In LinkedList we used get Entry to do the same thing. It will throw a warning when peekFront attempted on an empty Queue.

3. Create an implementation plan, listing out the order in which you plan to implement the classes. If some classes will be developed in parallel, discuss this.

StackInterface

Stack
QueueInterface
Queue
PreconditionViolationException
Executive
Node
main

4. After completing your implementation, describe in a one or more paragraphs what the most difficult part of the implementation for you was. There isn't a right or wrong answer to this. Please do discuss. We're interested in response like "The Stack class was hard."

I feel the Executive class is the hardest one, and especially “status”. Since the Theater class should not use the isEmpty method to check status of stack and queue, before making any calls to the class methods.
I was trying to add methods like getLength(), but adding methods is not allow too.