COMP132 – Computer Science II
Fall 2015

Homework #12
Stacks and Queues
Solutions

1. Consider each of the following real world applications. Indicate if the application would be better implemented using a stack or a queue. If the operations needed by the application do not match our Stack and Queue ADT operations exactly choose the one that most closely matches the needs.

a. A calendar program keeping a list of the upcoming events for which it needs to display alerts for the user (e.g. Dr. appt. today at 12:00!").

Queue: If events are entered into the Queue in the order in which they will occur the first element in the queue will always be the next event. Thus, the calendar must only look at the first element of the queue to determine when the next alert should be displayed.

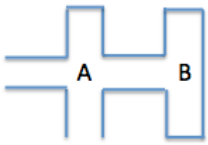
b. The discard pile for a card game in which each player may choose to take either a card from the deck or the card discarded by the previous player.

Stack: The player may only take the top card from the discard pile. If cards are added to the stack as they are discarded then the top card can be retrieved with a pop operation.

c. A program for finding a path through a maze. The program begins by noting all possible paths from the starting location (e.g. N, S, and W). It then follows one of the possible paths to the next intersection where it again notes all possible paths (perhaps S and E). If a dead-end is reached it goes back to the most recent intersection and tries one of the other paths. If all paths from an intersection have been tried, the program backtracks to the next previous intersection and tries a path from there that has not already been tried.

Stack: At each newly visited intersection all possible directions are pushed onto the stack, ensuring that the one from which you arrived is pushed first. The top direction is popped off of the stack and pursued. If that is a dead end, the next direction will be popped off of the stack and pursued.

For example, consider what would happen upon arriving at intersection A in the maze shown below:



Upon arriving at A, and assuming we push directions in a counter-clockwise order, the directions W, S, E, N would be pushed onto the stack. North would be popped off of the stack and explored. It is a dead-end, so the next direction would be popped off (E) and explored. Going east we arrive at location B and push W, S, N onto the stack. We pop N and find a dead-end, we pop S and find a dead-end, we pop-W and return to location A. Because we have already visited A, instead of pushing the directions, we pop S from the stack and explore in that direction. And so on. If the path to the S results in no solution, we would again return to A again, this time popping W from the stack and returning along that path to continue the search.

d. A web server that receives requests for web pages and then sends back the web page for each request in the order that they were received.

Queue: This is a first-in-first-out (FIFO) system. If each request is placed onto the tail of the queue as it is received, the next request to be processed will always be the one at the head of the queue.

e. A simulation program to determine which of the following strategies for processing customers at a bank is more efficient. The first strategy has all customers wait in a single line and when a teller opens the first customer in line goes to the open teller. In this system any teller can serve any customer. In the second strategy every teller has his or her own line. Customer's choose a line and wait for their turn for the associated teller.

Queue: Both of these systems are based on simulating customers waiting in a line for service. Customers are serviced in FIFO order in both models. Thus, in the first model a single queue can be used and all tellers remove customer's from the head of the queue when they are available. In the second model there will be one queue for each teller and each teller will only remove customer's from his/her own queue.

2. Each of the following snippets of code uses either a Stack or a Queue or both. Give the output that would be produced by each snippet.

a.
```
CS132Stack stk = new CS132LinkedStack();
stk.push("A");
stk.push("B");
System.out.println(stk.pop());
System.out.println(stk.peek());
stk.push("C");
System.out.println(stk.pop());
System.out.println(stk.pop());
```

```
B
A
C
A
```

b.
```
CS132Queue que = new CS132LinkedQueue();
que.add("X");
que.add("Y");
System.out.println(que.peek());
System.out.println(que.remove());
que.add("Q");
System.out.println(que.peek());
que.add("Z");
System.out.println(que.remove());
```

```
X
X
Y
Y
```

c.
```
CS132Stack stk = new CS132LinkedStack();
CS132Queue que = new CS132LinkedQueue();
stk.push("A");
stk.push("B");
stk.push("C");
que.add("X");
que.add("Y");
que.add("Z");
for(int i=0; i<4; i++) {
  que.add(stk.pop());
  stk.push(que.remove());
}
while(stk.size() > 0) {
  System.out.println(stk.pop());
}
```

```
C
B
A
```

3. If a sequence of objects are pushed onto a stack successive calls to pop will return the objects in the reverse of the order in which they were pushed. Consider the following method that might appear in the `CS132SinglyLinkedList` class from lab:

```
public void reverse()
```

Give an implementation of this method that uses a `Stack` to reverse the nodes in a `CS132SinglyLinkedList`. Bonus: Have your method work by directly manipulating the references contained in the `SinglyLinkedNodes` instead of by calling other methods in the `CS132SinglyLinkedList` class. (+2)

```
public void reverse() {
  CS132Stack stk = new CS132LinkedStack();

  // Push all of the nodes in the list onto the stack.
  SinglyLinkedNode cur = head;
  while(cur != null) {
    stk.push(cur);
    cur = cur.next;
  }

  /*
   * Make head the first thing off of the stack.
   * Note if the stack is empty then this will be null.
   */
  head = (SinglyLinkedNode)stk.pop();
  cur = head;

  /*
   * Pop the rest of the nodes off of the stack and link
   * them into the list as we go.
   */
  while(stk.size() != 0) {
    SinglyLinkedNode next = (SinglyLinkedNode)stk.pop();
    cur.next = next;
    cur = next;
  }

  /*
   * Make sure the next reference of the last node in
   * the list is null.
   */
     cur.next = null;
}
```

4. Give an implementation of the `CS132Queue` interface that uses a `CS132SinglyLinkedList` to hold the elements in the queue. Use an approach similar to what was used to create the `CS132LinkedStack` class.

NOTE: JavaDoc omitted for brevity.

```
public class CS132LinkedQueue implements CS132Queue {

  private CS132SinglyLinkedList elements;

  public CS132LinkedQueue() {
    elements = new CS132SinglyLinkedList();
  }

  public void add(Object obj) {
    elements.add(obj);
  }

  public Object peek() {
    if (elements.size() > 0) {
      return elements.get(0);
    }
    else {
      return null;
    }
  }

  public Object remove() {
    if (elements.size() > 0) {
      return elements.remove(0);
    }
    else {
      return null;
    }
  }

  public int size() {
    return elements.size();
  }
}
```