EECS 268: Spring 2009
Laboratory 7: Inheritance

Due: 11:59:59 pm the day before your lab meets the week of April 6th

Lecture Topics: Inheritance, Stacks, Queues
Lab Topics: Inheritance, Virtual Functions

1. Introduction

With the influx of monetary funds from moderately aggressive stock trading facilitated by your stock management software, your client, an independently wealthy Mary Kay consultant, has been able to acquire more merchandise to sell and thus increase her client base. This dramatic increase in people who depend on her for quality cosmetics at a reasonable price has bogged down her weekly distribution parties and as such is reducing customer satisfaction. She would like you to create a simple simulation system to give her insight into how these parties could be improved.

For this lab you will implement this simple simulation software that emulates the customer line of this distribution party. This line will encapsulate both a queue and a stack in its implementation. In order to run your simulation, you will read in commands from a file and perform actions based on these commands. Furthermore, inheritance will be needed, as there will be multiple types of customers being held in the stack and queue data structures.

2. Application Structure

Your simulation will reuse the stack and a queue implemented in the last lab to manage a series of customers in line to acquire beauty products. You will have 3 types of customers: generic customers, regular customers and important customers. The important customers and regular customers will be subclasses of the more general generic customer class.

Here is how your program will operate:

2.1 Command Processing

Your program will read in entries from a file, one line at a time. Here are the possible values the file could contain:

- customer-name
  - The customer's first name
  - Example: `Doris`
  - This customer will be added to the line based on rules described below
- `done`
  - The customer at the front of the line is finished buying and is removed from the line
  - If another customer exists, that customer begins to be served.
  - A print statement will indicate who is finished and who is currently being served.
- `print`
  - The name of the customer currently being served is output to standard out.

2.2 Stack and Queue Use

Each time a name is read from the file, a new customer is created and added to the line. Regular customers and Generic Customers will be added to the queue component of the line. Regular customers will have known names, such as 'Doris' or 'Samantha'. Generic customers come into play when your client can't quite remember

the name of a customer. Perhaps it's a new client or a friend of a friend. Whatever the case, those names that begin with the word 'Customer' will be represented as generic customers. They will go on the same queue as the regular customers. Examples of generic customer names would be 'Customer3' and 'Customer55'.

If the customer's name begins with the letter '**V**' (the secret-special-Kay consonant of the month) then this customer will be considered an "important customer" and immediately become the current customer being served and will not have to wait in line as normal, customers. When an individual with a V-name is read in, they will become instances of the 'important customer' class.

The secret-special-Kay consonant of the month comes with some strings attached. Courtesy is the name of the game for these select individuals. As such, if an important customer is being served and another important customer is read in from the file, then this newer important customer will become the current customer being served. Once the newer customer has been served, the previous V-named individual will return as being the current person being served. So, you will use a Stack to simulate this behavior. Each time a new important customer is read into the file, it will be added to the stack. This will allow the most recent important customer to be served first.

Each time the current customer being served changes (whether an existing customer completes, or a new customer usurps the front position) the customer will speak to the cosmetic distributer. What the customer says will depend on what type of customer they are, as detailed below.

2.3 Inheritance

As mentioned before, important customer and regular customer will subclass the generic customer class. The generic customer class will contain a 'talk' virtual method that will be overridden by its sub-classes. This method will return a string representing the conversation had between your client and the current customer. Here is a prototype for this method:

```
virtual std::string talk();
```

Each customer will say a different thing. Generic Customers will speak of the weather and their pets. Regular customers will output a simple greeting, an indication about what they are planning to buy, or blather on about where they plan to show off their new make-up and accessories. Special customers, on the other hand, will say coherent sentences having a majority of words that start with the letter V (as mandated by the Kay Grammar and Consultation Guide). To implement this, you will override the virtual talk method in the special customer class in order to change what they say.

3. Other Information

- There may be not enough 'done' commands to remove all the customers in the line. You are expected to remove these unaccounted for customers in the correct sequence before the termination of the program. One way to do this would be in the destructor of your Line class, that encapsulates the rest of the classes from the main function.
- There may be too many 'done' commands, in which case your program should be able to deal with this error by outputting an error message and continuing with the execution of the program.
- Your program should be able to handle a print command when no customers are present in the line.
- The customer instances don't necessarily need to say something different each time, but it wouldn't hurt.
- All error handling should be done utilizing exceptions.

4. Input & Output

4.1 Sample Input File

```
Customer1
Doris
Anna
Vicky
print
done
```

4.2 Sample Output File

```
Customer1 Says: “I love this season’s colors.  I’m more of an autumn
anyways.”
Vicky Says: “I vexed the vicar with my Viva Voce video’s volume very loud.”
Vicky is currently being served
Vicky is done, Customer1 is now being served.
Customer1 Says: “I love this season’s colors.  I’m more of an autumn
anyways.”
Customer1 is done, Doris is now being served.
Doris Says: “Thanks for special ordering that rose-petal pink blush, my
grandmother loved it!”
Doris is done, Anna is now being served.
Anna Says: “This new mineral powder and foundation set is great.”
Anna is done, no one else left in the line.
```

5. Grading Criteria

30	**Correct use of Inheritance**
20	**Correct implementation of Line class**
10	**Correct Stack / Queue Use**
15	**Exception / Error Handling**
10	**Documentation**
10	**Programming style**
5	**Correctness**