Project – Milestone 2

Assembly Line – Customer Orders

In this project, you are to code a simulation of an assembly line in three separate milestones.

LEARNING OUTCOMES

Upon successful completion of this project, you will have demonstrated the abilities to

- design and code a composition
- work with vector and queue containers from the Standard Template Library
- work with class variables and functions
- parse a string into tokens
- report and handle exceptions
- move objects from one container to another

SUBMISSION POLICY

Each milestone is to be submitted by its scheduled date. If you cannot complete a milestone by its scheduled date, ask your instructor for an extension. As a general rule, your instructor at their discretion will grant you one extension only for the entire project. A late submission of a milestone with no extension or a submission with a second or third extension will attract a late penalty. The late penalty is 20% of the project mark for each late submission.

In order to be eligible for a credit in this course, you need to submit a workable solution for this project. If you submit all the milestones except the last and have a sufficiently high grade to pass the course with a grade of 0 for the project, your instructor may enter an incomplete (INC) grade for you, in which case you will be required to submit a workable solution before the start of the next semester. If you do not submit a workable solution by the required date, you will fail the course.

Milestone Submission Dates:

1. Milestone 1 – Inventory Item Sets and Project Utilities – March 8 23:59
2. **Milestone 2 – Customer Orders – March 22 23:59**
3. Milestone 3 – Stations and Line Manager – April 5 23:59

PROJECT OVERVIEW

The project simulates an assembly line that fills customer orders from inventory. Each customer order consists of a list of items that need to be filled. The line consists of a set of stations. Each station holds an inventory of items for filling customer orders and a queue of orders to be filled. Each station fills the next order in the queue if that order requests its item and that item is still in stock. A line manager moves the customer orders from station to station until all orders have been processed. Any station that has used all of the items in stock cannot fill any more orders. Orders become incomplete due to a lack of inventory at one or more stations. At the end of all processing, the line manager lists the completed orders and the orders that are incomplete. The figure below illustrates the classes that constitute the simulator and the process of filling orders as they move along the pipeline.

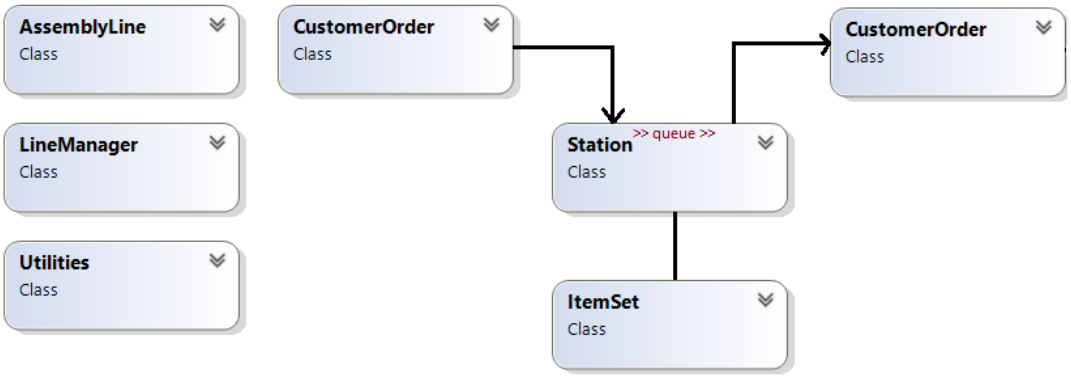


Figure 1 – Simulator Classes

SPECIFICATIONS

Milestone 2 builds the Customer Orders module for the Assembly Line. This milestone consists of five modules:

- **project** (supplied)
- **AssemblyLine** (supplied)
- **CustomerOrder**
- **Utilities** (from Milestone 1)
- **ItemSet** (from Milestone 1)

Enclose all your source code within the **sict namespace** and include the necessary guards in each header file. The output from your executable running Visual Studio with the following command line argument should look like

```
Command Line : C:\Users\...\Debug\MS2.exe Inventory.txt CustomerOrders.txt

Customer Order Assembly
```

```
======================

Items in Stock
--------------
CPU          [123456] Quantity 5   Description: Central Processing Unit
Memory       [654321] Quantity 10  Description: Basic Flash Memory
GPU          [456789] Quantity 2   Description: General Processing Unit
SSD          [987654] Quantity 5   Description: Solid State Drive
Power Supply [147852] Quantity 20  Description: Basic AC Power Supply

For Manual Validation: Item 1
 getName(): CPU
 getSerialNumber(): 123456
 getQuantity(): 5
 getSerialNumber(): 123457
 getQuantity(): 4

Customer Orders
---------------
Elliott C.   [Gaming PC]
             CPU
             Memory
             GPU
             GPU
             GPU
             SSD
             Power Supply
Chris S.     [Laptop]
             CPU
             Memory
             SSD
             Power Supply
Mary-Lynn M. [Desktop PC]
             CPU
             Memory
             Power Supply
Chris T.     [Micro Controller]
             GPU
             GPU
             Power Supply
             SSD

For Manual Validation
Chris T.     [Micro Controller]
             GPU
             GPU
             Power Supply
             SSD
Chris T.     [Micro Controller]
             GPU
             GPU
             Power Supply
             SSD

Chloe        [Flight PC]
             CPU
             GPU
             Power Supply
```

```
Mary-Lynn M. [Desktop PC]
             CPU
             Memory
             Power Supply

For Manual Validation Filling
Mary-Lynn M. [Desktop PC]
             CPU
             Memory
             Power Supply
 isFilled(): false
 Filled Mary-Lynn M. [Desktop PC][CPU][123457]
 isFilled(): false
 Unable to fill Mary-Lynn M. [Desktop PC][CPU][123457] already filled
 Filled Mary-Lynn M. [Desktop PC][Memory][654321]
 Filled Mary-Lynn M. [Desktop PC][Power Supply][147852]
 isFilled(): true

Customer Order Assembly Complete
```

The input for testing your solution is stored in two files `Inventory.txt` (from Milestone 1) and `CustomerOrders.txt`. The names of these files are specified on the command line as shown in red above. The records in the `CustomerOrders.txt` file are

```
Elliott C.|Gaming PC|CPU|Memory|GPU|GPU|GPU|SSD|Power Supply
Chris S.|Laptop|CPU|Memory|SSD|Power Supply
Mary-Lynn M.|Desktop PC|CPU|Memory|Power Supply
Chris T.|Micro Controller|GPU|GPU|Power Supply|SSD
```

Each record consists of no less than 3 fields delimited by a prescribed char ('|') for the project. These fields are:

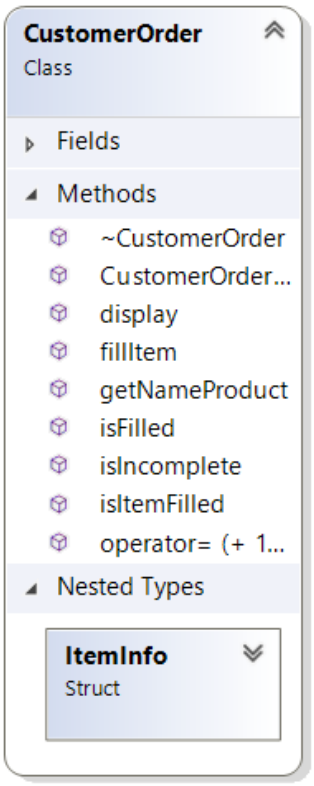
- Name of the customer
- Name of the product being assembled
- Names of the items to be added to the product

Customer Order Module

The Customer Order module contains all the functionality for handling customer orders as they move along the assembly line. As the line manager moves an order along the assembly line, the station where that order currently rests fills a request for an item of that station, if there is any such request. Once the order has reached the end of the line, the order is either completed or incomplete. One cause of incompleteness is the lack of sufficient items in stock at a station.

Design and code a class named **CustomerOrder** for managing and processing customer orders. Customer order objects are unique and hence neither copyable nor copy-assignable. However, they are both movable and move-assignable.

Your class design includes the following public member functions:



- A default constructor that sets the object to a safe empty state.
- A one-argument constructor that receives a reference to an unmodifiable string. This constructor extracts no less than 3 tokens from the string. The first extracted token holds the customer's name. The second token holds the name of the product being assembled. The remaining tokens hold the names of the items to be added to the product throughout the assembly process. This function throws an exception if no items have been requested to be added; that is, there are less than 3 token in the string. Otherwise, this function allocates memory for each item to be added with its fulfillment information. (Hint: one way to hold this information is in array of **ItemInfo** sub-objects nested within the object itself. Information that needs to be tracked includes the name of the item, its serial number, and its filled status.) This constructor determines the field width to be used in displaying customer names for all orders managed by this class.
- A destructor that deallocates any memory that the object has allocated.
- `void fillItem (ItemSet& item, std::ostream& os)` – a modifier that receives a reference to an `ItemSet` object and an `std::ostream` object. This function checks each item request, fills it if the requested item is available and the request has not been filled, reports the filling in the format shown below and decrements the item stock by one:

  Filled CUSTOMER [PRODUCT][ITEM][SERIAL NUMBER]

  If the item request has already been filled or if the item is out of stock, this function displays the corresponding message:

  Unable to fill CUSTOMER [PRODUCT][ITEM][SERIAL NUMBER] already filled

  Unable to fill CUSTOMER [PRODUCT][ITEM][SERIAL NUMBER] out of stock

- `bool isFilled() const` – a query that searches the list of items requested and returns true if all have been filled; false otherwise.
- `bool isItemFilled(const std::string& item) const` – a query that receives the name of an item, search the item request list for that item and returns true if all requests for that item have been filled; false, otherwise. If the item is not in the request list, this function returns true.
- `std::string getNameProduct() const` – a query that returns the name of the customer and their product in the following format:

CUSTOMER [PRODUCT]

- **`void display(std::ostream& os, bool showDetail) const`** – a query that receives a reference to an **std::ostream** object **os** and a Boolean **showDetail** and inserts the data for the current object into stream **os**. If the Boolean is **false** the data consists of the name of the customer, the product being assembled, and the list of items on the order.

  CUSTOMER [PRODUCT]
  ITEM
  ITEM
  ITEM

  Otherwise, the data consists of the name of the customer, the product being assembled, and the list of items with detail information on the order. Details include name of the item, its serial number and its filled status.

  CUSTOMER [PRODUCT]
  [SERIAL NUMBER] ITEM – FILL_STATUS
  [SERIAL NUMBER] ITEM – FILL_STATUS
  [SERIAL NUMBER] ITEM – FILL_STATUS

  All item information is indented exactly the number of spaces required for the longest customer name + 1. See the output above for an example of the formatting.

Milestone Submission

To test and demonstrate execution of your program use the same data as shown in the output example above.

Upload your source code to your `matrix` account. Compile and run your code using the latest version of the gcc compiler and make sure that everything works properly.

Then, run the following command from your account: (replace `profname.proflastname` with your professor's Seneca userid)

~profname.proflastname/submit 345XXX_ms2<ENTER>

and follow the instructions. Replace **XXX** with the section letter(s) specified by your instructor.