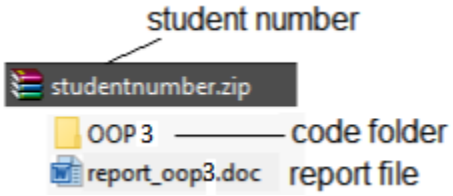
Programming Assignment 3

Sales Order Application

Instructions:

1. Write an application to implement the "**Sales Order System**" as described below.
2. The solution is to be presented in a **report**. This report is to contain
   - A through description of the application,
   - Classes Diagram,
   - A description of underlying data structures used to solve the problem,
   - A copy of the C++ code,
   - A description of all the function/routines which have been used, and
   - Test data used and sample execution screenshots of output produced.
3. A copy of the code and an executable is to be placed in a folder called **OOP3**. A copy of the code must also be uploaded on teams with the following structure.



4. You will be required to demo your application for assessment after the due date.
5. **Due Date (Over Teams/Drive)**: 23th April 2022 @11:59PM

Components:

1: The Stock

The stock class will store the list of products with the ability to add/edit and delete products. There is a count value that returns the number of products. Also there is a search function to look up on a product and return the quantity of product if found. Operators overload for reading and printing stock data in the system.

2: The Customer /Customers

The customer can be a person or a customer. For any, one can add/edit and delete customers from the system. Operators overload for reading and printing a customer. And print function for print all customers in the system

3: The Product

The product class contains the product information. On can update product information. The operators overload for reading and printing a product.

4: The Order

The Order class contains the order number (random int from 1 to 100000), order date, and total amount of order. The create order for creating an order, update order to update order status. Order status can be one of the following states (new, hold, paid and canceled). Edit order to change order information. And operators overload for reading and printing an order.

5: The OrderItem

The order item class holds item sale price and product quantity (the order quantity must be less than or equal quantity of product in stock). And Operator overloading (++,--,+=-,-=) for increase/decrease the quantity or order. Any change of quantity must follow the stock requirement.

Operator	Calculate Payroll
++	Increase quantity by 1
--	Decrease quantity by 1
+=n	Increase quantity by n
-=n	Decrease quantity by n

6: Payment

The payment can be credit, cash or check. The Payment date and amount. All child classes must add additional information needed.

7: Transaction

The transaction class holds orders and records payment. Operator += for add new transaction with payment for an order and recorded in the system. Operators overload << for printing all transactions.

8: Demo

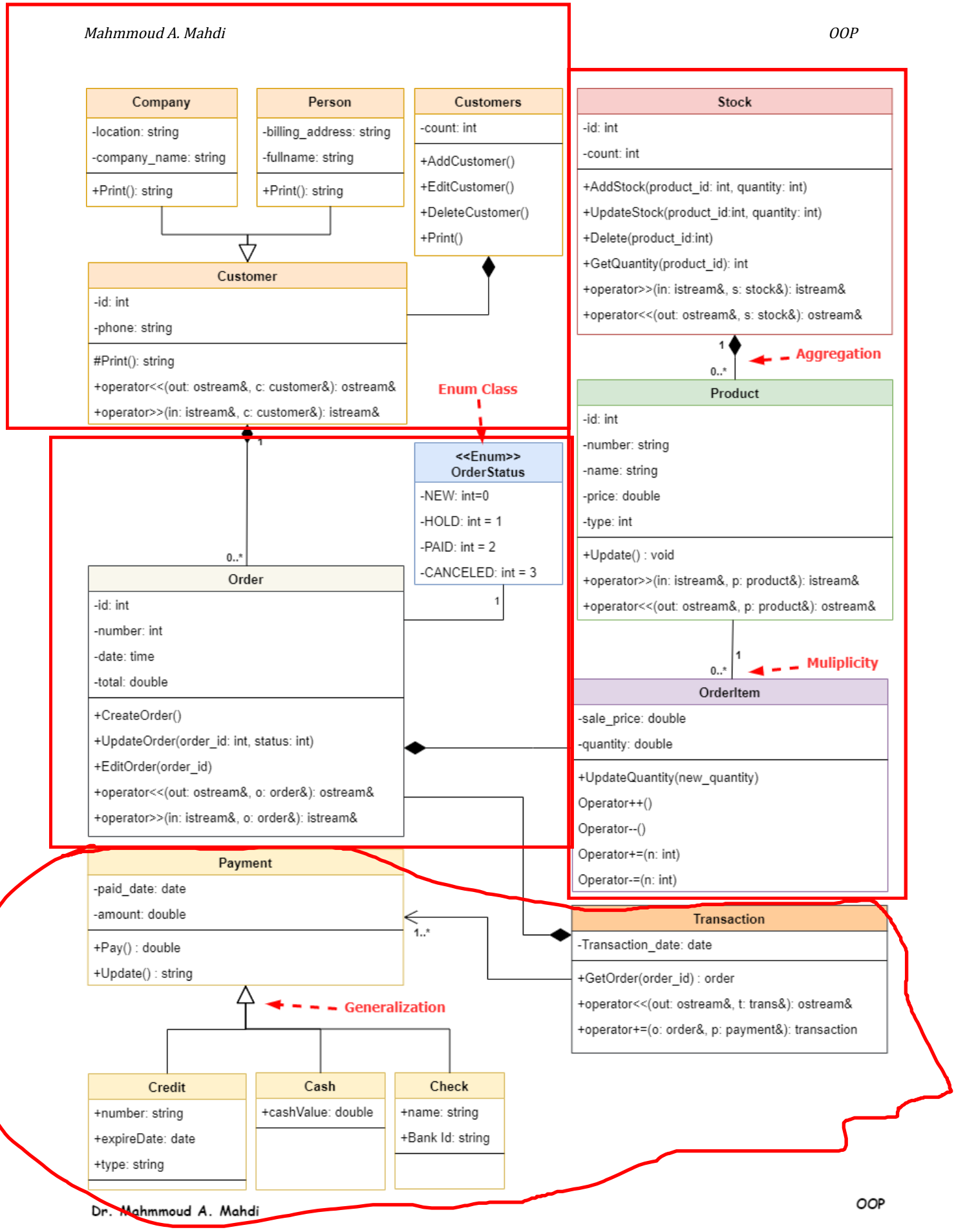
The main function should follow the following ***minimum*** menu:

1. **Data Entry**
    1. Add New/Update/Delete Customer
    2. Add New/Update/Delete Product in Stock
2. **Sales Process**
    1. Add Transaction
        -> Enter Customer: .....
        -> Enter Items: .....
        -> Enter Status: …
    2. Update Order
        -> Order item Quantity
        -> Order Status
    3. Pay Order
3. **Print**
    1. Customers
    2. Stock data
    3. Transactions

NOTE:

Add all requirement functions you may need such as get/set or other.

Starter Class Diagram


Company
-location: string
-company_name: string
+Print(): string
Person
-billing_address: string
-fullname: string
+Print(): string
Customers
-count: int
+AddCustomer()
+EditCustomer()
+DeleteCustomer()
+Print()
Customer
-id: int
-phone: string
#Print(): string
+operator<<(out: ostream&, c: customer&): ostream&
+operator>>(in: istream&, c: customer&): istream&
Stock
-id: int
-count: int
+AddStock(product_id: int, quantity: int)
+UpdateStock(product_id:int, quantity: int)
+Delete(product_id:int)
+GetQuantity(product_id): int
+operator>>(in: istream&, s: stock&): istream&
+operator<<(out: ostream&, s: stock&): ostream&
1
0..*
Aggregation
Enum Class
Product
-id: int
-number: string
-name: string
-price: double
-type: int
+Update() : void
+operator>>(in: istream&, p: product&): istream&
+operator<<(out: ostream&, p: product&): ostream&
<<Enum>>
OrderStatus
-NEW: int=0
-HOLD: int = 1
-PAID: int = 2
-CANCELED: int = 3
1
0..*
Order
-id: int
-number: int
-date: time
-total: double
+CreateOrder()
+UpdateOrder(order_id: int, status: int)
+EditOrder(order_id)
+operator<<(out: ostream&, o: order&): ostream&
+operator>>(in: istream&, o: order&): istream&
1
0..*
Muliplicity
OrderItem
-sale_price: double
-quantity: double
+UpdateQuantity(new_quantity)
Operator++()
Operator--()
Operator+=(n: int)
Operator-=(n: int)
Payment
-paid_date: date
-amount: double
+Pay() : double
+Update() : string
1..*
Transaction
-Transaction_date: date
+GetOrder(order_id) : order
+operator<<(out: ostream&, t: trans&): ostream&
+operator+=(o: order&, p: payment&): transaction
Generalization
Credit
+number: string
+expireDate: date
+type: string
Cash
+cashValue: double
Check
+name: string
+Bank Id: string