Minions Ordering Food

Start Assignment

- Due Dec 11 by 11:59pm
- Points 160
- Submitting a file upload
- File Types cpp and h
- Available Aug 8 at 8am - Dec 11 at 11:59pm

Food Ordering System

Stuart, Kevin, and Bob have arrived at New York and they have discovered New York food! No more bananas for them. They want boba teas and food.

Unfortunately, they can't dine in at the restaurants due to COVID. They decided to order food from their phone using this food ordering system that you are writing for them.

Exception

The system will have one exception class *InvalidInput*. Initialize the constant class member *message* with the following(*input* is a string parameter passed into the constructor):

```
"Invalid input \"" + input + "\".\n"
```

Please initialize the private constant string variable *message* with **initialization list** on the constructor of the exception.

The exception class has one void function *reason()* that will output the *message*.

Given that the exception class is short, you don't need a .cpp file for implementation. You can have all implementation located in the .h file.

Classes

You will have a total of four classes for this system. You need to a header file and a source file for each of the class. The four classes include

- **abstract** class *DeliveryOrder*

- class *BobaOrder* inherited from the DeliveryOrder class
- class *FoodOrder* inherited from the DeliveryOrder class
- class *Account*

1. DeliveryOrder

Class DeliveryOrder will have the following member attributes:

- *name*(string - customer's name)
- *date*(string - date of the order)
- *phone*(string - phone number of the customer)
- *miles* (float - number of miles for the delivery)
- *orderBalance*(float - balance of the order which doesn't include delivery fee and tax)

The class DeliveryOrder also has three **static** members including *orderCount*, which is used to calculate how many orders are placed, and two **constant** variables *taxRate* and *deliveryRate***.**

All of these attributes are **private** to class *DeliveryOrder*, except the *orderBalance* which should **only** be directly accessible by the inherited classes (protected).

Class DeliveryOrder will have a **constructor**, a **destructor**, and **four functions:**

- The constructor will initialize customer's name, date of the order (month/day/year), phone number, and number of miles of the delivery from the parameters. It will initialize the orderBalance to be **0**, then increase the orderCount so that we can count how many orders are created
- The destructor will output a string "DeliveryOrder destroyed.\n".
- The four functions include
  - **constant** *receipt()* function that will print out the order receipt
  - **constant** *getTotalBalance()* function that will return the total balance(with delivery fee and tax)
  - **static** *getOrderCount()* function that will return the *orderCount*
  - **pure virtual** function *VIPdiscount()* that will calculate and return the discount for the order

Here I've provided the content of the receipt() function for you so you don't need to worry about the output format.

```
cout << "Order Detail:" << "\n";
cout << "\tName: " << name << "\n";
cout << "\tDate: " << date << "\n";
cout << "\tPhone: " << phone << "\n";
cout << "\tOrder Balance: $" << orderBalance << endl;
```

The *getTotalBalance()* function should calculate the total balance by adding the tax and delivery fee to the *orderBalance*. Here's the formula:

```
orderBalance * (1 + taxRate) + miles * deliveryRate;
```

Make sure you initialize the static variables *taxRate* to be **0.09** and the *deliveryRate* to be **$2** for the class. Initialization of static variables need to be written in the source file.

2. BobaOrder

BobaOrder should inherit from DeliveryOrder with **public inheritance**. It has two additional member attributes, *shopName* and *drinksCount*. The constructor of BobaOrder should takes the same parameters as DeliveryOrder's constructor and an extra string to initialize the member attribute shopName. The destructor will simply output

```
"BobaOrder destroyed.\n"
```

BobaOrder will override the *receipt()* function by adding one extra line to the receipt as following:

```
cout << "\tDrinks Count: " << drinksCount << endl;
```

Make sure you **call the receipt() function** of the DeliveryOrder class instead of rewriting the same code again.

You will also override the *VIPdiscount()* function as it's required for pure virtual functions. The discount function will

- return 0.8 if the number of drinks is greater than 10
- return 0.9 if the number of drinks is greater than 5
- return 0.95 if the number of drinks is greater than 2
- return 1 if the number of drinks is less than or equal to 2

BobaOrder also has a new function *addDrink()* which will take in the name of the drink, a boolean default to true indicating whether one wants to add boba to the drink, and a count default to 1 indicating how many of the same drink one wants to order. There are three drinks available for people to order from:

- Green Tea Latte: $5.8
- Brown Sugar Boba Milk: $7.8
- Brown Sugar Pearl Milk: $9.8

You will do string comparison to match the drinks with the parameter. If the passed in drink doesn't match any of the available drinks, throw the *InvalidInput* exception and pass in the drink name to the exception. Adding boba will cost **$1** per drink, and make sure you multiply the passed in count in case one wants to order more than one of the same drink. For example, if Kevin orders two Green Tea Latte with bobas, the cost will be **($5.8 + $1) * 2**.

At the end of the function make sure you add the number of drinks to the *drinksCount* and add the cost of this drink order to the *orderBalance*.

3. Class FoodOrder

FoodOrder is similar to *BobaOrder* that it inherits from *DeliveryOrder* with public inheritance. It also has two additional member attributes, *restaurantName* and *foodCount*. The constructor of *FoodOrder* takes the same parameters as DeliveryOrder's constructor and an extra string to initialize the new member *restaurantName*. The destructor will simply output

```
"FoodOrder destroyed.\n"
```

Similarly to *BobaOrder*, *FoodOrder* will override the *receipt()* function by adding one extra line to the receipt as following:

```
cout << "\tFood Count: " << foodCount << endl;
```

Make sure you call the *receipt()* function of the DeliveryOrder class instead of rewriting the same code again.

FoodOrder will also override the *VIPdiscount()* function as it's required for pure virtual functions. The discount function will

- return 0.8 if the *orderBalance* is greater than 50
- return 0.9 if the *orderBalance* is greater than 30
- return 0.95 if the *orderBalance* is greater than 20
- return 1 if the *orderBalance* is less than or equal to 20

Remember that the value *orderBalance* doesn't include the tax or the shipping cost.

FoodOrder also has a new function *addFood()* which will take in the name of the main course ordered, an integer **default to 0** indicating how many sides one wants to order, and a boolean **default to false** indicating whether one wants to add soup to the order. There are four meals available for people to order from:

- Thick Cauliflower Steaks: $15
- Rack of Lamb: $38
- Organic Scottish Salmon: $23
- Grilled Lobster Risotto: $46

You will do string comparison to match the meal and if the passed in meal doesn't match any of the meal listed on the menu, throw the *InvalidInput* exception with meal name as the parameter. Adding soup costs **$5**, and each additional side costs **$6**. For example, if Bob orders the Rack of Lamb with one additional side and soup, the cost will be **$38 + $5 + $6**.

At the end of the function make sure you update the *foodCount* and add the cost of this food order to the *orderBalance* of the order.

4. Account

The account class will have two private attributes: *username* and *status*. The constructor will take in a string to initialize the username and another string default to "Regular" to initialize the status. The status

is optional but can be "VIP", "Owner", or "Regular". The destructor will output

```
"Account removed.\n"
```

We will have one getter function *getStatus* to return the status of the account. Remember all non-static getter functions should be constant.

Function

We will have one non-member function for this program. The function is *applyDiscount()* which will take in a *DeliveryOrder* **pointer** and a **constant reference** *account* and returns a float. The function will calculate the discount then apply the discount on the total balance(returned from function `getTotalBalance()`) of the order. For example, if the total balance is $100 and the discount is 0.1, then this function will return a float 10.0 which is the result of (100 * 0.1).

The *applyDiscount()* function will check the account status to determine how to apply the discount.

- If the account status is "Owner", it will apply 90% off to the order. Therefore, it will return 0.1 multiply by the total balance.
- If the account status is "VIP", it will trigger the corresponding *VIPdiscount()* function based on the order with dynamic binding. It will then take the returned value of *VIPdiscount()* and multiply with total balance then return the result.
- If the account status is "Regular", no discount will be applied and the delivery balance of the order will remain unchanged and returned.

Main()

Finally, it's our main function. We first have **three accounts**, one constant owner account, one VIP account, and one regular account.

- Stuart appears to be the secret owner of all the restaurants and boba shops of New York, he will have an owner account the system created for him and he can't change the account, so it's a **constant** account. The account username will be "Stuart" and the status will be "Owner".
- Kevin appears to be the secret VIP of all the restaurants and boba shops of New York, and he created his VIP account with username "Kevin" and status "VIP".
- Bob appears to be a regular customer, who's not aware of Kevin and Stuart's secret identities, created his regular account with username "Bob". (You don't pass in the status when creating Bob's account because it should be an optional parameter)

We will then have a **DeliveryOrder pointer** that we will later use to point to different orders to allow polymorphism. It's ok for your new pointer not to point at anything when it's created.

1. Kevin Placing Order

Kevin starts to order boba drinks for everyone. Let's first output

```
"Kevin is placing order.\n"
```

He creates his order with his name "Kevin", date "04/20/2024", phone number "123-456-0000". He's placing order at the shop "M Tea" and looks like the miles for the delivery is 10.4 miles.

Kevin adds his first drink order "Green Tea Latte" with all default choices for this drink, then his second drink "Brown Sugar Pearl Milk" with no boba, and two more "Brown Sugar Boba Milk" with no boba. Kevin added another order "Iron Goddess" without knowing that the shop isn't serving this drink anymore. This should return an error message for him.

For this assignment, you can put **all the addDrink() function calls in one single try catch block**. Catch the InvalidInput exception by reference and trigger the *reason()* function of the exception. Then output

```
"Not serving requested drinks. Drink order ignored.\n\n"
```

before we end the catch block.

Now the system should print out his receipt using the *receipt()* function. Make sure you set *cout* to only print out **2 decimal places**. Then we output the delivery balance by calling the *getTotalBalance()* function. The format should be similar to "Balance: $99.99\n". Don't forget the **dollar sign**. Then we want to output the discounted balance like this "Discounted Balance: $88.88\n". We get the discounted balance by having our **DeliveryOrder pointer points to Kevin's order** and trigger the *applyDiscount()* function with Kevin's account. Finally we finished the receipt by having two addition new lines printed at the end. (Detail output format see the expected output below)

2. Stuart Placing Order

Stuart then wants to order foods for everyone. Let's again output

```
"Stuart is placing order.\n"
```

Stuart creates his order with name "Stuart" on date "04/20/2024". His phone number is "123-456-1111". He's ordering from the restaurant "Tavern Green" which is 25.5 miles from their place. He's ordering a "Thick Cauliflower Steaks" with one side and one soup, a "Organic Scottish Salmon" with no side or soup, and a "Rack of Lamb" with one soup but no side.

Similarly here that you can put all the addFood() function calls in a single try catch block for this assignment. Catch the *InvalidInput* exception by reference and trigger the reason() function of the exception. Then output

```
"Not serving requested food. Food order ignored.\n\n"
```

before we end the catch block.

Now the system print out his receipt with *receipt()* then output the delivery balance by calling *getTotalBalance()* function. Then we output the discounted balance by calling the *applyDiscount()* function. Make sure you use the same format as indicated above. And we finish the print statements with two extra new lines.

3. Bob Placing Order

Now Bob found out that Stuart get such good pricing on food ordering, he's wondering whether he can get that too. Let's output

```
"Bob decided to log in to his account and see whether he can afford ordering the same order as Stuart.\n"
```

Bob is trying to place the same order, so he's using the **same order object** that Stuart created.

We start by printing the receipt of the order. Again, you can call the *receipt()* function from the order Stuart created. Then we output the delivery balance as before by calling *getTotalBalance()* function. Followed by printing out the discounted balance but this time we **pass in Bob's account** to the *applyDiscount()* function. Keep the same format as before when you print. Bob sees the discounted balance and he's upset that he needs to pay so much. Therefore he decided **not to** place the order and have Stuart do it. Therefore the system will simply output

```
"Bob upset, cancelling order :(\n\n"
```

Finally, we will output the number of order placed. The format will be

```
"Total order placed: 2.\n\n"
```

You get the number of order placed by calling the **static function** *getOrderCount()* **from DeliveryOrder**.

Expected Output

Here's the expected output.

```
Kevin is placing order.
Invalid input "Iron Goddess".

Not serving requested drinks. Drink order ignored.

Order Detail:
        Name: Kevin
        Date: 04/20/2024
        Phone: 123-456-0000
        Order Balance: $32.20
        Drinks Count: 4
Balance: $55.90
Discounted Balance: $53.10


Stuart is placing order.
Order Detail:
        Name: Stuart
        Date: 04/20/2024
```

```
        Phone: 123-456-1111
        Order Balance: $92.00
        Food Count: 3
Balance: $151.28
Discounted Balance: $15.13


Bob decided to log in to his account and see whether he can afford ordering the same order as Stuart.
Order Detail:
        Name: Stuart
        Date: 04/20/2024
        Phone: 123-456-1111
        Order Balance: $92.00
        Food Count: 3
Balance: $151.28
Discounted Balance: $151.28
Bob upset, cancelling order :(

Total order placed: 2

FoodOrder destroyed.
DeliveryOrder destroyed.
BobaOrder destroyed.
DeliveryOrder destroyed.
Account removed.
Account removed.
Account removed.
```

Submission

You will submit a header file and a source file for each of the class and a main.cpp file to house the applyDiscount function and main function. Please include proper documentation as we've been doing for all other projects. Make sure you name your functions as required and only use what we've covered in class to finish this assignment. Leveraging online solution will results in 0 credit.

Rubric

5 pts	Exception class constructor const message initialized correctly
5 pts	reason() reason() output message correctly
5 pts	DeliveryOrder constructor Initialized all variables and increment count correctly
2 pts	DeliveryOrder destructor Output correctly
5 pts	DeliveryOrder::VIPdiscount() Correct syntax to define it as a pure virtual function
2 pts	DeliveryOrder::getOrderCount() Static getter function to get the order count

5 pts	DeliveryOrder::receipt() constant function and print out receipt correctly
2 pts	DeliveryOrder::getTotalBalance() constant function and return the balance correctly
5 pts	DeliveryOrder variables static constant variables tax rate, delivery rate, and static count initialized and defined correctly. Protected variables and private variables defined correctly.
5 pts	BobaOrder constructor Call DeliveryOrder constructor and initialize correctly
2 pts	BobaOrder destructor Output correctly
2 pts	BobaOrder new variables Two private variables shopName and drinksCount
2 pts	BobaOrder::addDrink() parameters BobaOrder::addDrink() pass in default parameters
5 pts	BobaOrder::addDrink() logic Implement drink price logic correctly Add to drink count and balance correctly
3 pts	BobaOrder::addDrink() error BobaOrder::addDrink() throw error correctly
5 pts	BobaOrder::VIPdiscount() Override discount correctly
5 pts	BobaOrder::receipt() override receipt by calling base function and add to output
5 pts	FoodOrder constructor Call DeliveryOrder constructor and initialize correctly
2 pts	FoodOrder destructor Output correctly
2 pts	FoodOrder new variables Two private variables restaurantName and foodCount
2 pts	FoodOrder::addFood() default variables

	FoodOrder::addFood() pass in default parameters
5 pts	FoodOrder::addFood() logic Implement food price logic correctly Add to food count and balance correctly
3 pts	FoodOrder::addFood() error FoodOrder::addFood() throw error correctly
5 pts	FoodOrder::VIPdiscount() Override discount correctly
5 pts	FoodOrder::receipt() override receipt by calling base function and add to output
5 pts	Account constructor Default parameter initialized correctly Initialize class variables correctly
2 pts	Account destructor Output correctly
2 pts	Account variables Two private variables
2 pts	Account::getStatus() constant and return correctly
10 pts	applyDiscount() function Takes in pointer and constant reference Implement logic correctly Return the correct value
6 pts	Main(): accounts Constant account created for Stuart VIP account created for Kevin Regular account created for Bob with only one parameter
2 pts	Main(): order pointer Create a pointer that will be pointing to different orders.
2 pts	Main() Kevin created first order object Pass in parameters correctly
4 pts	Kevin addDrink() Added four drinks with the right parameters passed in.

3 pts	Catch Kevin exception Try all add drink function call and throw error by reference. Print out reason and messages. Ignore error and continue
2 pts	Kevin receipt() Print out receipt correctly
4 pts	Kevin discounted balance Call applyDiscount with the order pointer pointing to Kevin's order. Return the correct discounted price.
2 pts	Stuart created second order object Pass in parameters correctly
3 pts	Stuart addFood() Added three food orders with the right parameters passed in.
3 pts	Catch Stuart exception Try all add food function call and throw error by reference. Print out reason and messages. Ignore error and continue
2 pts	Stuart receipt() Print out receipt correctly
4 pts	Stuart discounted balance Call applyDiscount with the order pointer pointing to Stuart's order. Return the correct discounted price.
2 pts	Bob receipt() Print out the food order receipt correctly
4 pts	Bob discounted balance Call applyDiscount with the order pointer pointing to Stuart's order and pass in Bob's account. Return the correct discounted price.
2 pts	Main() output order count Output the count of orders correctly.