

# Minions Ordering Food

[Start Assignment](#)

---

**Due** May 18 by 11:59pm    **Points** 160    **Submitting** a file upload    **File Types** cpp and h  
**Available** Apr 28 at 9pm - May 18 at 11:59pm 20 days

---

## 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 class member **message** with a string parameter **input** as following:

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

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

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



## 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 an **abstract** class **DeliveryOrder**, two **concrete** classes **BobaOrder** and **FoodOrder** inherited from the DeliveryOrder class, and one more class **Account**.

### 1. DeliveryOrder

Class DeliveryOrder will have the following member attributes: customer's **name**(string), **date** of the order(string), **phone** number(string), number of **miles** for the delivery(float), and **orderBalance**(float, doesn't include delivery fee and tax). The class DeliveryOrder also has three **static** members including the **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". Given that we have a **virtual** function in this class, our destructor should also be **virtual**.
- The four functions include a **constant receipt()** function that will print out the order receipt, a **constant getTotalBalance()** function that will return the total balance(with delivery fee and tax), a **static getOrderCount()** function that will return the `orderCount`, and a **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 delivery order balance by adding the tax and delivery fee. Here's the formula:

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

Make sure you initialize the static variables `taxRate` to be **0.0887** 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 piece of code again.


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

- return 0.85 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: \$6.6
- Brown Sugar Boba Milk: \$8.8
- Brown Sugar Pearl Milk: \$9.9

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 passed in the drink name as s. Adding boba will cost **\$1.5** 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 2 Green Tea Latte with bobas,

 cost will be  **$(\$6.6 + \$1.5) * 2$** .

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

### 3. 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 also takes in an extra string to initialize the member attribute 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 piece of code again.

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

- return 0.85 if the `orderBalance`(without tax and shipping cost) is greater than 50
- return 0.9 if the `orderBalance`(without tax and shipping cost) is greater than 30
- return 0.95 if the `orderBalance`(without tax and shipping cost) is greater than 20
- return 1 if the `orderBalance`(without tax and shipping cost) is less than or equal to 20

`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 Steak: \$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 **\$6**, and each additional side costs **\$5**. 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 a 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 getter functions should be constant.

## Function


We will have one function for this system. The function is **applyDiscount()** which will take in a **DeliveryOrder pointer** and a **constant reference account**. The function will return the discounted balance of the order. For example, if the total balance(returned from function `getTotalBalance()`) is \$100 and the discount is 0.1, then this function will return the float 10 which is  $(100 * 0.1)$ .

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, so it will return 0.1 multiply by the delivery balance
- If the account status is "VIP", it will trigger the corresponding **VIPdiscount()** function of the order depending on whether it's a **BobaOrder** or a **FoodOrder** with dynamic binding. It will then take the returned value of `VIPdiscount()` and multiply with delivery 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 Function

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".

We will then have a **DeliveryOrder pointer** that we will later use to point to different orders to allow polymorphism.

### 1. Kevin Placing Order

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

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

He creates his order with his name "Kevin", date "1/20/2022", phone number "123-456-0000". He's placing order at the shop "Tiger Sugar" 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.

Because each of the addDrink() function could cause exception, let's put all the addDrink() function calls in a try catch block. Catch the InvalidInput exception by reference and trigger the print\_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. We first need to call printReceipt() function to print out the receipt. 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. (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 placing order.\n" first.

Stuart creates his order with name "Stuart", same date as Kevin's order. His phone number is "123-456-1111". He's ordering from the urant "Tavern Green" which is 25.5 miles from their place. He's ordering a "Thick Cauliflower Steaks" with one sides and soup, a "Organic Scottish Salmon" with no side no soup, and a "Rack of Lamb" with no sides and soup.

Similarly here that each of the addFood() function could cause exception, let's put all the addFood() function calls in a try catch block. Catch the InvalidInput exception by reference and trigger the print\_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. Similarly we first call printReceipt() to print out the receipt for his order. Then we output the delivery balance by calling getBalance() function. Same format as indicated above. Then we output the discounted balance using the same format

as above as well. You would need to get the discounted balance by calling the `applyDiscount()` function as well. And we finish the receipt 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 can start by printing the receipt for Bob. We can directly call the `printReceipt()` function from the order Stuart created. Then we output the delivery balance as before by calling `getBalance()` 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 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 `getOrderCount()` function from `DeliveryOrder`.

## Expected Output

Here's the expected output.



```
Kevin placing order.  
Invalid input "Iron Goddess".  
Not serving requested drinks. Drink order ignored.  
  
Order Detail:  
  Name: Kevin  
  Date: 1/20/2022  
  Phone: 123-456-0000  
  Order Balance: $35.60  
  Drinks Count: 4  
Balance: $59.56  
Discounted Balance: $56.58  
  
Stuart placing order.  
Order Detail:  
  Name: Stuart
```



```
Date: 1/20/2022
Phone: 123-456-1111
Order Balance: $105.00
Food Count: 3
Balance: $165.31
Discounted Balance: $16.53
```

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: 1/20/2022
Phone: 123-456-1111
Order Balance: $105.00
Food Count: 3
Balance: $165.31
Discounted Balance: $165.31
Bob upset, cancelling order :(
```

Total order placed: 2


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


## Submission





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.


## Final Project Rubric


Criteria	Ratings		Pts
Exception class constructor const message initialized correctly	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
print_reason() output message correctly	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
DeliveryOrder constructor Initialized all variables and increment count correctly	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
DeliveryOrder destructor Virtual and output correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
 DeliveryOrder::VIPdiscount() Correct implementation and is pure virtual	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
DeliveryOrder::getOrderCount() Static getter function to get the order count	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts


Criteria	Ratings		Pts
DeliveryOrder::receipt() constant function and print out receipt correctly	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
DeliveryOrder::getTotalBalance() constant function and return the balance correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
DeliveryOrder variables static constant variables tax rate, delivery rate, and static count initialized and defined correctly. Protected variables and private variables defined correctly.	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
BobaOrder constructor Call DeliveryOrder constructor and initialize correctly	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
 BobaOrder destructor Output correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
BobaOrder new variables Two private variables shopName and drinksCount	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts

Criteria	Ratings		Pts
BobaOrder::addDrink() pass in default parameters	2 pts Full Marks	0 pts No Marks	2 pts
BobaOrder::addDrink() logic Implement drink price logic correctly Add to drink count and balance correctly	5 pts Full Marks	0 pts No Marks	5 pts
BobaOrder::addDrink() throw error correctly	3 pts Full Marks	0 pts No Marks	3 pts
BobaOrder::VIPdiscount() Override discount correctly	5 pts Full Marks	0 pts No Marks	5 pts
 BobaOrder::receipt() Override receipt by calling base function and add to output	5 pts Full Marks	0 pts No Marks	5 pts
FoodOrder constructor Call DeliveryOrder constructor and initialize correctly	5 pts Full Marks	0 pts No Marks	5 pts

Criteria	Ratings		Pts
FoodOrder destructor Output correctly	2 pts Full Marks	0 pts No Marks	2 pts
FoodOrder new variables Two private variables restaurantName and foodCount	2 pts Full Marks	0 pts No Marks	2 pts
FoodOrder::addFood() pass in default parameters	2 pts Full Marks	0 pts No Marks	2 pts
FoodOrder::addFood() logic Implement food price logic correctly Add to food count and balance correctly	5 pts Full Marks	0 pts No Marks	5 pts
 FoodOrder::addFood() throw error correctly	3 pts Full Marks	0 pts No Marks	3 pts
FoodOrder::VIPdiscount() Override discount correctly	5 pts Full Marks	0 pts No Marks	5 pts

Criteria	Ratings		Pts
FoodOrder::receipt() override receipt by calling base function and add to output	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
Account constructor Default parameter initialized correctly Initialize class variables correctly	<b>5 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	5 pts
Account destructor Output correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
Account variables Two private variables	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
 Account::getStatus() constant and return correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
applyDiscount() function Takes in pointer and constant reference Implement logic correctly Return the correct value	<b>10 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	10 pts

Criteria	Ratings		Pts
Main() create accounts Constant account created for Stuart VIP account created for Kevin Regular account created for Bob with only one parameter	<b>6 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	6 pts
Main() created order pointer	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
Main() Kevin created first order object Pass in parameters correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
Kevin addDrink() Added four drinks with the right parameters passed in.	<b>4 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	4 pts
 h Kevin exception Try all add drink function call and throw error by reference. Print out reasons and messages. Ignore error and continue	<b>3 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	3 pts
Kevin receipt() Print out receipt correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts

Criteria	Ratings		Pts
Kevin discounted balance Call applyDiscount with the order pointer pointing to Kevin's order. Return the correct discounted price.	<b>4 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	4 pts
Main() Stuart created second order object Pass in parameters correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
Stuart addFood() Added three food orders with the right parameters passed in.	<b>3 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	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	<b>3 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	3 pts
 Print receipt() Print out receipt correctly	<b>2 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	2 pts
Stuart discounted balance Call applyDiscount with the order pointer pointing to Stuart's order. Return the correct discounted price.	<b>4 pts</b> <b>Full Marks</b>	<b>0 pts</b> <b>No Marks</b>	4 pts



Criteria	Ratings		Pts
Bob receipt() Print out the food order receipt correctly	2 pts Full Marks	0 pts No Marks	2 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.	4 pts Full Marks	0 pts No Marks	4 pts
Main() output order count Output the count of orders correctly.	2 pts Full Marks	0 pts No Marks	2 pts
Total Points: 160			

