CPSC 4620 Project Phase 4

Using a program to access your database

100 pts

Due: Tuesday, June 18th at 11:59 pm

For this stage of the project you will complete an application that will interact with your database. Much of the code is already completed, and you will just need to handle the functionality to retrieve information from your database and save information to the database. Not all of the functionality from the original prompt will be there, as that would be a lot to ask for in a limited time frame. Because of this, not every piece of information in your database will be needed. For instance, we will not calculate the cost to the business of the pizza. It would not be difficult to add this functionality, just time consuming.

Also note that this program does little to no verification of the input that is entered in the interface or passed to the objects in constructors or setters. This means that any junk data you enter will not be caught and will propagate to your database if it does not cause an exception before then. Be careful with what you enter as data. In the real world this program would be much more robust, and also much more complex.

Program Requirements:

1. Add a new order to the database: You must be able to add a new order with pizzas to the database. The user interface is there to handle all of the input, and it creates the `Order` object in the code. It then calls `DBNinja.addOrder(order)` to save the order to the database. That function is what you will need to complete. Remember that will include not just adding the order, but the pizzas and their toppings as well. Since you are adding a new order, the inventory level for any toppings used will need to be updated. You do not need to check to see if there is inventory available for all of the ingredients before the order is placed. You can just let the inventory level go negative for this project. In order to complete this, `DBNinja` must also be able to return a list of the available toppings and the list of known customers.
2. Enter a new customer: The program must be able to add the information for the new customer in the database. Again, the user interface for this exists, and it creates the `Customer` object and passes it to `DBNinja` to be saved to the database. You just need to add the code to add this customer to the database. **You do need to edit the prompt for the user interface in `Menu.java` to specify the format for the phone number, to make sure it matches the format in your database.**
3. View current orders: The program must be able to display all currently open orders. An order is open if the pizzas on the order have not been marked as complete. Again, the user interface exists for this, it just needs the functionality in `DBNinja`
4. Mark an order as completed: Once the kitchen has finished prepping an order, they need to be able to mark it as completed. When an order is marked as completed, all of the pizzas should be

marked as completed in the database. Again, the user interface exists for this, it just needs the functionality in `DBNinja`

5. View Inventory Levels: This option will display each topping and it's current inventory level. Again, the user interface exists for this, it just needs the functionality in `DBNinja`
6. Add Inventory: When the inventory level of an item runs low, the restaurant will restock that item. When they do so, they need to enter into the inventory how much of that item was added. They will select a topping and then say how many units were added. Note: this is not creating a new topping, just updating the inventory level. Again, the user interface exists for this, it just needs the functionality in `DBNinja`
7. Your code needs to be secure, so any time you are adding any sort of parameter to your query that is a `String` you need to use `PreparedStatements` to prevent against SQL injections attacks. If your query does not involve any parameters, or if your queries parameters are not coming from a string variable, then you can use a regular `Statement` instead.

The Files:

Start by downloading the starter code files from Canvas. You will see that the user interface and the java interfaces and classes that you need for the assignment are already completed. Review all these files to familiarize yourself with them. They contain comments with instructions for what to complete. You should not need to change the user interface except to change prompts to the user to specify data formats (i.e. dashes in phone number) so it matches your database. You also should not need to change the entity object code, unless you want to remove any ID fields that you did not add to your database. If you have any data types that don't match (i.e. string size options as 1, 2, 3 and 4 instead of strings), make the conversion when you pull the information from the database or add it to the database. You need to handle data type differences at that time anyways, so it makes sense to do it then instead of making changes to all of the files to handle the different data type or format.

The `Menu.java` class contains the actual user interface. This code will present the user with a menu of options, gather the necessary inputs, create the objects, and call the necessary functions in `DBNinja`. Again, you will not need to make changes to this file except to change the prompt to tell me what format you expect the phone number in (with or without dashes).

There is also a static class called `DBNinja`. This will be the actual class that connects to the database. This is where most of the work will be done. You will need to complete the methods to accomplish the tasks specified.

Also in `DBNinja`, there are several public static strings for different crusts, sizes and order types. By defining these in one place and always using those strings we can ensure consistency in our data and in our comparisons. You don't want to have "SMALL" "small" "Small" and "Personal" in your database so it is important to stay consistent. These strings will help with that. You can change what these strings say in `DBNinja` to match your database, as all other code refers to these public static strings.

Start by changing the class attributes in `DBNinja` that contain the data to connect to the database. You will need to provide your database name, username and password. All of this is available by visiting buffet.cs.clemson.edu. Once you have that done, you can begin to build the functions that will interact with the database.

The functions you will need to complete are already defined in the `DBNinja` class and are called by `Menu.java`, they just need the code. Two functions are completed (`getInventory` and `getTopping`), although for a different database design, and is left to show an example of connecting and using a database. You will still need to make changes to these functions to get them to work for your database.

Several extra functions are suggested in the `DBNinja` class. It is not required to complete these functions, but their functionality will be needed in other functions. By separating them out you can keep your code modular and reduce repeated code. I recommend completing your code with these small individual functions and queries. There are also additional functions suggested in the comments, but without the function template that could be helpful for your program.

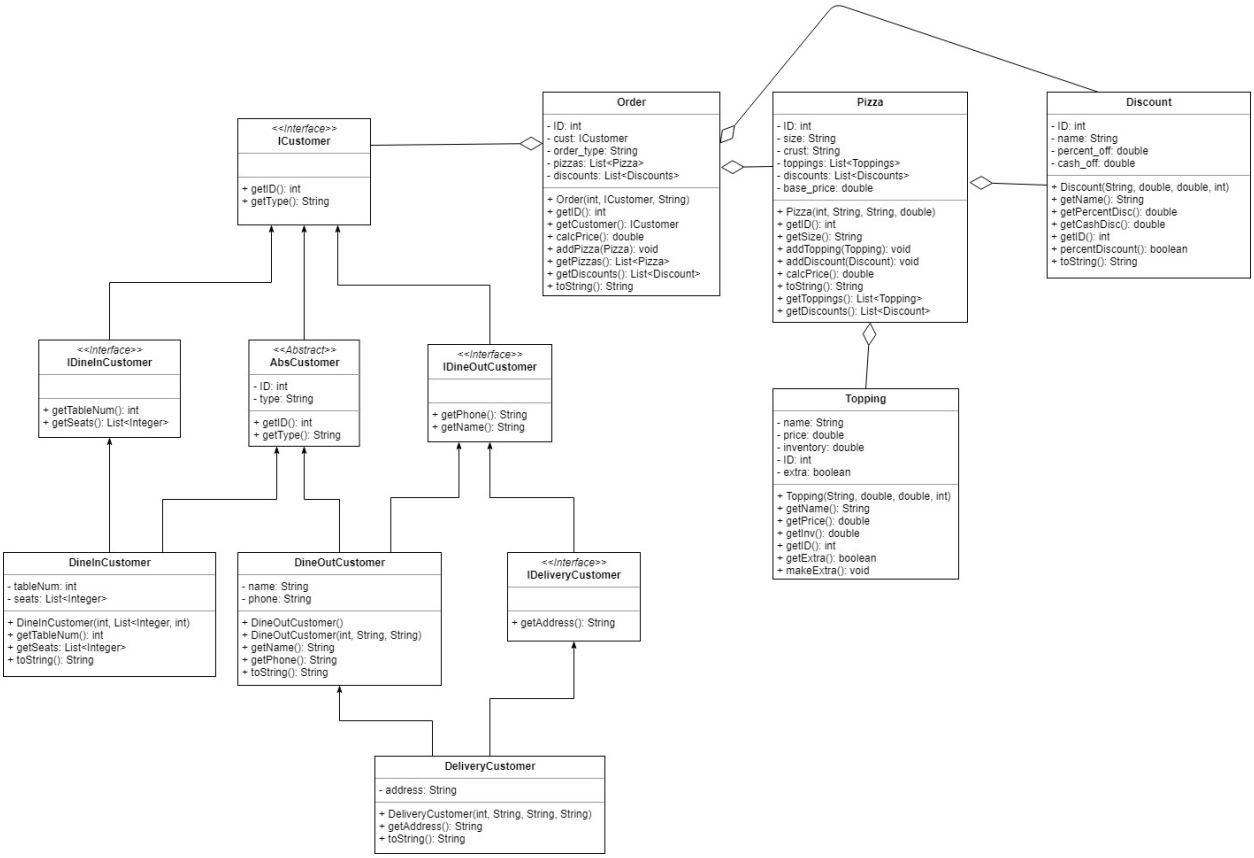
If the code in the `DBNinja` class is completed correctly, then the program should function as intended. Make sure to try it out to ensure your code is functioning.

Additionally, in the starter code there is a directory called "lib." Leave that directory exactly as it is. It contains the drivers necessary to connect to the MySQL database.

The other files:
There are also several files that you do not need to (and should not) edit at all, but you still need to be familiar with them to use them in your `DBNinja` class. Many of them are straightforward entity objects (Pizza, topping, discount, order), but the representation for `Customer` gets a little trickier. There is an `ICustomer` interface, with different interfaces that extend it: `IDineInCustomer`, `IDineOutCustomer`. Additionally, `IDineOutCustomer` is extended by `IDeliveryCustomer` to add in the address for the customer. `ICustomer` is implemented by `AbsCustomer` which is abstract and cannot be instantiated directly. `AbsCustomer` is extended by `DineInCustomer` and `DineOutCustomer`. `DineOutCustomer` is also extended by `DeliveryCustomer`. Our system follows our best practice of coding to the interface, so we always use `ICustomer` as the declared type since every type of customer implements `ICustomer`. When working with your database, you'll have to carefully choose which customer type to use for each customer.

Make sure to familiarize yourself with all of these files, and get an idea of the design of this system before working on your own code. You'll also notice that I included the contracts for these classes and methods, which are there to help you understand the system and how to use the objects. Don't worry if you never call some of the methods. They may just be there for the user interface.

The following UML diagram will (hopefully) help you understand the structure of the system:



Compiling and running your code:

Your code also contains the makefile you need to compile your code. You can compile your code by using the command “make” (while in the StarterCode directory), then run the code using “make run”. Use those commands, and do not try to compile on your own, as you need to include the paths to the MySQL driver.

The code that I have provided will compile, it just will not function without your additions. Because so much code is being provided, there is no excuse for turning in code that will not compile on one of the Clemson unix machines. Code that does not compile on a Clemson unix machine will receive a 0, even if the issue is minor and easy to correct.

Submission

You will submit your assignment on Handin. Make sure to include all the code files and the make files and zip them up in one compressed file to keep the file structure. You do not need to submit the lib directory.

Your code will be connecting to your database on the mysql server. Make sure not to go and make changes to your database that could affect the functionality of the program.

No late submissions will be accepted for this assignment.

Notes and Tips:

Recall how casting and determining dynamic types works in java. Consider a function that takes in an `ICustomer` object and we need to know whether it is a dine in, delivery, or pickup customer so we know what methods to call:

```
void foo(ICustomer cust)
{

	if(cust instanceof DeliveryCustomer)
	{
		//is a delivery customer
		//need to cast
		DeliveryCustomer c = (DeliveryCustomer) cust;
		//now can call DeliveryCustomer methods on c
		c.getAddress();
	}
	elseif(cust instanceof DineOutCustomer)
	{
		//is a pickup customer
		//need to cast
		DineOutCustomer c = (DineOutCustomer) cust;
		//now can call DineOutCustomer methods on c
		c.getPhone();
	}
	elseif(cust instanceof DineInCustomer)
	{
		//is a delivery customer
		//need to cast
		DineInCustomer c = (DineInCustomer) cust;
		//now can call DineInCustomer methods on c
		c.getTable();
	}

}
```

FAQ:

Q: Can I run the code locally on my own machine?

A: It is possible to set up, but I don't know the steps for the process. Students have done so in the past, but they also told me it was a huge hassle to set up. I recommend just running on the school unix

machines. Note: recently we started blocking Microsoft Secure Shell (it was very insecure) but PuTTY and WinSCP both still work with the school machines.

Q: The objects in the code (Pizzas, orders, customers) all come in with IDs of -1. How do I assign an ID to it for the database?
A. Before adding to the database you will need to find a valid ID to use for your database. How you do this depends on your database, but if you are using a numeric value, the easiest way is to query the maximum ID value, then add 1 to it for your new ID.

Q: There is no code to calculate the business cost. Do I have to do that myself?
A: For simplicity sake I left that functionality out of this stage of the project. Feel free just to insert a random cost ($2.00 or something) for each new pizza to make it work for now.

Q: Does GetCustomer List need to return DineIn customers?
A. No, just delivery and pickup

Q: I don't use an ID Number as my primary key. What should I do?
A. You can ignore that field. Just pick a random number to use as the ID in the constructor (not -1). You can also go delete all references to it, but you shouldn't need to.

Q: I don't have a DineInCustomer table, I just store the table and seat numbers with the DineInOrder. Do I need to change my database?
A: No. Remember the way data is stored in the database will not match the way it is stored in the program. You will still use the DineInCustomer class (as order does not have a field for table number and seats), but you will pull that information from (or add it to) your table that holds dine in orders.

Q: Your helper functions don't have any parameters! How can I implement them?
A: Add parameters. I left the parameters blank because I didn't know what your primary key was. They are not called anywhere, just suggestions that would make your code simpler.