CPSC 4620/6620 Project:
Part 3 – Java Application

Due: Tuesday April 23 @ 11:59 pm

100 pts

For this part of the project, you will complete an application that will interact with your database. You will implement the functionality to create, retrieve, update and delete information in your database (also known as a CRUD application).

Note, the starter code for this application has little to no verification of the input that is entered in the interface or passed to the objects in the constructors and setters. This means that any junk data you enter will not be caught but will propagate to your database if it does not cause an exception. **Be careful** with the data you enter! In the real world, this program would have a much better data entry system.

IMPORTANT NOTE: You will be given starter code (explained in detail below). You can complete the project by making changes to 3 files: `Menu.java, DBNinja.java` and `DBConnector.java`. You can add additional methods to these classes, but do NOT delete any of the methods in the starter code OR change their method definitions (ie the names or data types of the parameters). Work within the framework given; being a competent software professional means you can work with code you haven't written (even if you don't like it). One specific Java note, not only is it bad programming practice to use multiple input objects in Java, it is a security risk and will break the Gradescope autograder. There is one `public BufferedReader` object created in the main class (`Menu.java`). Use it for all your input operations. Failure to do so will force Gradescope to give you a 0 for any automated tests that require input.

Program Requirements:

1. **Add a new order to the database:** You must be able to add a new order with pizzas to the database. Use a combination of print statements and the `BufferedReader` reader object provided to ask the user of your program for input. Remember, there's more to adding an order to the DB than just creating an order and sending it to the database via SQL. You need to create all the pizzas that go in the order, update topping inventories as necessary, checking to make sure a topping inventory never goes negative, and apply discounts to pizzas and orders. This means you'll be updating more than just the order table for this operation. Also, you can assume that a topping can only be added to a pizza 1 time.
2. **View customers:** This option will display each customer and their associated information. The customer information must be ordered by last name, first name and phone number. Most of the functionality for this will be done in the `DBNinja` file and you will just call the appropriate method(s) from the `Menu` class.
3. **Enter a new customer:** The program must be able to add the information for a new customer and store it in the database. Be sure you prompt the user for all the necessary information **(phone numbers should be entered as ##########, 10-digits with no punctuation).**

4. **View orders:** The program must be able to display orders sorted by order date/time from most recent to oldest. The program must display:
    a. all open orders
    b. all completed orders
    c. all orders (open and completed) since a specific date (inclusive)
5. **View order details**: This option is part of the "**View orders**" option defined in #4. After displaying the list of orders as specified in "**View orders**" above, the program must allow the user to select a specific order for viewing its details. The details include the full order type information, the pizza information (including pizza discounts), and the order discounts.
6. **Mark an order as completed:** Once the kitchen has finished prepping an order, they need to be able to mark it as completed. The program must display the open orders sorted by order date/time and allow the user to select the order to "complete."
7. **View Inventory Levels:** This option will display each topping and its current inventory level. The toppings must be displayed sorted in alphabetical order.
8. **Add Inventory:** When the inventory level of an item runs low, the restaurant will restock that item. When they do so, they need to increase the inventory by a specified amount. The program needs to display the list of toppings and allow the user to select a topping and then specify how many units to add. Note: this is not creating a new topping, just updating the inventory level of an existing topping. Make sure that the inventory list is sorted in alphabetical order when displayed.
9. **View Reports:** The program must be able to select and display each of the 3 profitability reports using the views created in Part 2. Specifically
    (a) ToppingPopularity
    (b) ProfitByPizza
    (c) ProfitByOrderType
10. Modify the package `DBConnector` to contain <u>your</u> database connection information, this is the same information you use to connect to the database via MySQL Workbench. You will use `DBNinja.connect_to_db method` to open a connection to the database. Be aware of how many open database connections you make and make sure the database is properly closed when the application exits! The grading will **NOT** be done using your database. A new database will be created using your SQL scripts and a new `DBConnector` class generated to connect to that database. ***THEREFORE, the scripts you submit with your code must match what the application is expecting! Otherwise, your application is guaranteed to fail!!***
11. Your code needs to be secure, so any time you are adding any sort of parameter to a 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.
12. There are several helper methods in `DBNinja` with names like "get…" and "find…", **all of these methods MUST be implemented**, even if you choose not to use them. Basically, if it's in `DBNinja`, then you need to implement the method.
13. Watch the Pizza Application video in Canvas. `Menu.java` has all the user prompts you need to match the output in the video. **You must make sure your input/output matches the video**. Adding, removing or reordering the expected input/output will guarantee the autograder will fail! Remember to use the `BufferedReader` object created in the main class (`Menu`) for all your input operations.

14. Regarding Java: Use Java version 8 and do NOT remove the `package cpsc4620` from your code, all of your code must reside in the `cpsc4620` package.

The Files:

Start by downloading the starter code from Canvas. The object classes (i.e., `Pizza.java`, `Order.java, etc.`) are already completed and should not be modified. The files you will be editing are `Menu.java, DBNinja.java` and `DBConnector.java` (to add your DB connection information). These files contain comments with instructions for what each method should do. You must use all the class files (i.e., it's not acceptable to delete the `DineinOrder`, `DeliveryOrder`, and `PickupOrder` class files and pretend they don't exist). You can add methods to the provided classes, but do not remove or alter the definitions of the existing methods.

The `Menu.java` is where your user interface should go. This code will present the user with a menu of options, gather the necessary inputs, create objects, and update the database using the methods in `DBNinja`. All the user prompts are included in `Menu.java`, you should use the Pizza Application video in Canvas as your guide on how to use the prompts.

The static class `DBNinja` is where most of the work will be done, the code written in the `Menu` class should be I/O operations and calls to `DBNinja`. You will need to complete ALL the methods in `DBNinja` to accomplish the tasks specified above. There should be no input operations in `DBNinja.` `DBNinja` also defines several public static strings for different crusts, sizes and order types. By defining these in one place and always using those strings you can ensure consistency in your data and in our comparisons. You don't want to have "SMALL", "small", or "Small" in your database, so it is important to stay consistent. These strings will help with that. You should **not** change these values.

Start by changing the class attributes in `DBConnector` that contain the data to connect to the database. You will need to provide your server name, database name, username, and password. All of this is covered in the Chapter 15 lecture materials and should be familiar from completing Part 2 of the project. Once you have that done, you can begin to build the methods that will interact with the database.

The methods you need to complete are already defined in `DBNinja`; they just need the code. The `getCustomerName` method in `DBNinja` is already completed for you as an example.

Keep your code modular by creating helper methods in `DBNinja`; by separating them out you can keep your code compact and reduce repeated code. HINT, make sure you test your SQL queries in MySQL Workbench BEFORE implementing them in code…it will save you a lot of debugging time!

If the code in `Menu` and `DBNinja` is completed correctly, then the program should function as intended. Make sure to TEST, to ensure your code works!

Remember that you will need to include the MySQL JDBC libraries when building this application. Otherwise, you will NOT be able to connect to your database.

Calculating prices:

If a discount is applied to a pizza that discount reduces the cost to the customer but does not impact the cost to the business. Once all the pizza prices are calculated and discounts applied, this can be rolled up to calculate the order price and costs and any order discounts can be applied to the entire order. The same discount can be applied to multiple pizzas and likewise the same discounts can also be applied to the order!

The other files:

The other files, namely the object classes, are there to save you time. Many of them are straightforward entity objects (pizza, topping, discount, customer), but the representation for Order gets a little trickier. Java is where we implement our subtypes. Since order is a supertype in our ERDs, the three subtypes (dineinOrder, pickupOrder, and deliveryOrder) all **extend** order. Refer to *inheritance in Java* for what that entails if you are unfamiliar with Java. When working with your database, you'll have to carefully identify which type each order is, because simply making everything an Order object will mean you lose access to the type and all the variables unique to each type (tableNum, address, etc.).

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. Don't worry if you never call some of the methods. They may just be there to support one particular database design, which means your design might not require the usage of those methods.

Submission

You will submit your assignment to Gradescope. Your submission must include:

- **Updated** DB scripts from Part 2 (all 5 scripts, even if some of them are unchanged)
  in a folder named: sql.
- All of the .java application files (even files that have not changed)
  in a folder named: cpsc4620.
- **Zip** the DB Scripts and the .java files (i.e. the application) into one compressed **ZIP** file. No other formats will be accepted. Do not submit the lib directory or an IntellJ or other IDE project, **just the code**.

You can upload as many times as you like to see the autograder results. FYI, not all the test cases will be visible to you**. Test your program carefully before submitting!** The autograder results will be used in conjunction with a manual review of your code to produce the final grade.

Late work:

No late submissions will be accepted for this assignment.

CHEATING:

Don't do it! The only way to learn the material is to do your own work…so DO IT! You are expected to do your own work, possibly in collaboration with ONE partner. Plagiarism detection will be run on the submitted solutions.

Getting Help:

Use MS Teams and/or Outlook to ask questions. **Do not wait until the last day to ask questions or get started!**

Notes and Tips:

Compiling and running your code:

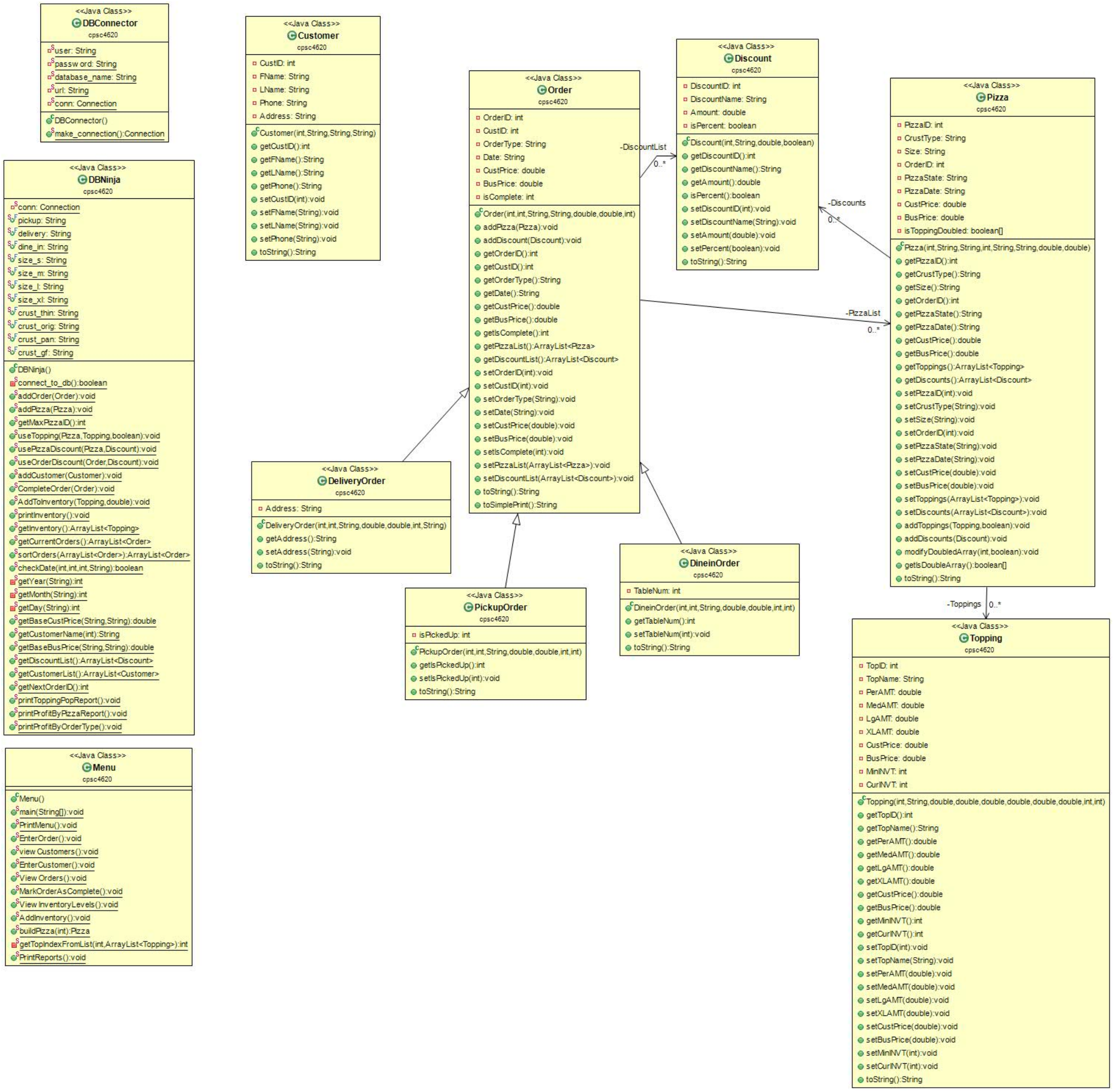
The starter code will compile and “run” as is, but it will not do anything useful without your additions. Because so much code is being provided, there is no excuse for submitting code that does not compile. Note that the autograder uses a unit test file that calls individual methods in DBNinja, so if you change the name, method signature, or delete a method your code will not compile. **Code that does not compile will receive severe deductions, even if the issue is minor and easy to correct.**

Note that your code will be tested on a server running Ubuntu, so pay careful attention to the case of your file names! The autograder shows you the exact input it uses for the I/O tests. Use this information to test your code.

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

```
void foo(Order o)
{
      if(o instanceof DeliveryOrder)
      {
            //is a delivery order -> need to cast
            DeliveryOrder c = (DeliveryOrder) o;
            //now can call DeliveryOrder method on o
            o.getAddress()
      }
}
```

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



<<Java Class>>
DBConnector
cpsc4620
user: String
passw ord: String
database_name: String
url: String
conn: Connection
DBConnector()
make_connection():Connection
<<Java Class>>
DBNinja
cpsc4620
conn: Connection
pickup: String
delivery: String
dine_in: String
size_s: String
size_m: String
size_l: String
size_xl: String
crust_thin: String
crust_orig: String
crust_pan: String
crust_gf: String
DBNinja()
connect_to_db():boolean
addOrder(Order):void
addPizza(Pizza):void
getMaxPizzaID():int
useTopping(Pizza,Topping,boolean):void
usePizzaDiscount(Pizza,Discount):void
useOrderDiscount(Order,Discount):void
addCustomer(Customer):void
CompleteOrder(Order):void
AddToInventory(Topping,double):void
printInventory():void
getInventory():ArrayList<Topping>
getCurrentOrders():ArrayList<Order>
sortOrders(ArrayList<Order>):ArrayList<Order>
checkDate(int,int,int,String):boolean
getYear(String):int
getMonth(String):int
getDay(String):int
getBaseCustPrice(String,String):double
getCustomerName(int):String
getBaseBusPrice(String,String):double
getDiscountList():ArrayList<Discount>
getCustomerList():ArrayList<Customer>
getNextOrderID():int
printToppingPopReport():void
printProfitByPizzaReport():void
printProfitByOrderType():void
<<Java Class>>
Menu
cpsc4620
Menu()
main(String[]):void
PrintMenu():void
EnterOrder():void
viewCustomers():void
EnterCustomer():void
ViewOrders():void
MarkOrderAsComplete():void
ViewInventoryLevels():void
AddInventory():void
buildPizza(int):Pizza
getTopIndexFromList(int,ArrayList<Topping>):int
PrintReports():void
<<Java Class>>
Customer
cpsc4620
CustID: int
FName: String
LName: String
Phone: String
Address: String
Customer(int,String,String,String)
getCustID():int
getFName():String
getLName():String
getPhone():String
setCustID(int):void
setFName(String):void
setLName(String):void
setPhone(String):void
toString():String
<<Java Class>>
Order
cpsc4620
OrderID: int
CustID: int
OrderType: String
Date: String
CustPrice: double
BusPrice: double
isComplete: int
Order(int,int,String,String,double,double,int)
addPizza(Pizza):void
addDiscount(Discount):void
getOrderID():int
getCustID():int
getOrderType():String
getDate():String
getCustPrice():double
getBusPrice():double
getIsComplete():int
getPizzaList():ArrayList<Pizza>
getDiscountList():ArrayList<Discount>
setOrderID(int):void
setCustID(int):void
setOrderType(String):void
setDate(String):void
setCustPrice(double):void
setBusPrice(double):void
setIsComplete(int):void
setPizzaList(ArrayList<Pizza>):void
setDiscountList(ArrayList<Discount>):void
toString():String
toSimplePrint():String
-DiscountList
0..*
<<Java Class>>
Discount
cpsc4620
DiscountID: int
DiscountName: String
Amount: double
isPercent: boolean
Discount(int,String,double,boolean)
getDiscountID():int
getDiscountName():String
getAmount():double
isPercent():boolean
setDiscountID(int):void
setDiscountName(String):void
setAmount(double):void
setPercent(boolean):void
toString():String
-Discounts
0..*
-PizzaList
0..*
<<Java Class>>
Pizza
cpsc4620
PizzaID: int
CrustType: String
Size: String
OrderID: int
PizzaState: String
PizzaDate: String
CustPrice: double
BusPrice: double
isToppingDoubled: boolean[]
Pizza(int,String,String,int,String,String,double,double)
getPizzaID():int
getCrustType():String
getSize():String
getOrderID():int
getPizzaState():String
getPizzaDate():String
getCustPrice():double
getBusPrice():double
getToppings():ArrayList<Topping>
getDiscounts():ArrayList<Discount>
setPizzaID(int):void
setCrustType(String):void
setSize(String):void
setOrderID(int):void
setPizzaState(String):void
setPizzaDate(String):void
setCustPrice(double):void
setBusPrice(double):void
setToppings(ArrayList<Topping>):void
setDiscounts(ArrayList<Discount>):void
addToppings(Topping,boolean):void
addDiscounts(Discount):void
modifyDoubledArray(int,boolean):void
getIsDoubleArray():boolean[]
toString():String
-Toppings
0..*
<<Java Class>>
DeliveryOrder
cpsc4620
Address: String
DeliveryOrder(int,int,String,double,double,int,String)
getAddress():String
setAddress(String):void
toString():String
<<Java Class>>
PickupOrder
cpsc4620
isPickedUp: int
PickupOrder(int,int,String,double,double,int,int)
getIsPickedUp():int
setIsPickedUp(int):void
toString():String
<<Java Class>>
DineinOrder
cpsc4620
TableNum: int
DineinOrder(int,int,String,double,double,int,int)
getTableNum():int
setTableNum(int):void
toString():String
<<Java Class>>
Topping
cpsc4620
TopID: int
TopName: String
PerAMT: double
MedAMT: double
LgAMT: double
XLAMT: double
CustPrice: double
BusPrice: double
MinINVT: int
CurINVT: int
Topping(int,String,double,double,double,double,double,double,int,int)
getTopID():int
getTopName():String
getPerAMT():double
getMedAMT():double
getLgAMT():double
getXLAMT():double
getCustPrice():double
getBusPrice():double
getMinINVT():int
getCurINVT():int
setTopID(int):void
setTopName(String):void
setPerAMT(double):void
setMedAMT(double):void
setLgAMT(double):void
setXLAMT(double):void
setCustPrice(double):void
setBusPrice(double):void
setMinINVT(int):void
setCurINVT(int):void
toString():String

FAQ:

Q: Can I start from scratch?

A: No. No. **No**.
Use the starter code. Learning how to work with someone else's code is a vital part of being a software developer.

Q: Do I have to use Java?
A: Yes…even if you've never used Java. You should have the skills necessary to pick up and understand a new computer language. If not, then you shouldn't be taking this class!

Q: Can I change the functions/methods provided in the files I am allowed to edit (`Menu.java, DBNinja.java, and DBConnector.java`)

A: No. You cannot delete or change the method definition of anything in these 3 classes. However, you can add methods as needed. The project can be completed with the methods as defined. There are comments in the starter code to help with your implementation.

Q: Will I have to make modifications to my part 1 and part 2 of the project for part 3?

A: Part 1? No. Part 2? Probably. Look at the object classes (or the UML diagram above) and you should be able to get a decent understanding of what the database that was used to make the starter code looks like. If there's an instance variable in the object class, it *usually* means that it's an attribute in the SQL table. Hopefully you've been making the suggested changes in the feedback on canvas to make this as smooth as possible.

Q: Can you provide the ERD for the database that was used to make the starter code?
A: No.

Q: What do I need to do before I submit?

A: **Test your program.** You should take advantage of the autograder in Gradescope. The autograder will be most helpful if you've done your own testing. If your code doesn't compile and run in the autograder, there will be extensive deductions. As noted above, you can submit as often as you like…but the autograder may restrict when and how much feedback it provides.

Q: How am I supposed to format my outputs?

A: Watch the Pizza Application video in Canvas. `Menu.java` has all the user prompts you need to match the output in the video**. You must make sure your input/output matches the video**. Adding, removing or reordering the expected input/output will cause the autograder to fail! Remember to use the `BufferedReader` object created in the main class (`Menu`) for all your input operations.

Q: How does partial credit work?

A: That's on a case-by-case basis. Your program should be complete and functional when you submit. If you're missing functionality to, say, print one of the three view reports, that's relatively minor. However, if your application is unable to make or add pizzas to the pizza database then we have nothing to grade. Nothing to grade means no points.

Q: So, if my program is unable to add orders, I don't get *any* credit?

A: Zero is such a small number, but if you cannot insert into the database, then what database functionality would work? Even if you wrote 20,000 lines of code, if we can't enter anything into your database, then your program does not work! If it can add orders, even if that order is full of incorrect information, that's when we get into partial credit territory.

Q: Do we have to use GitHub?

A: While not required, it's highly encouraged. GitHub makes it easy to backtrack when needed.

Q: What happens if I cannot get the connection set up?

A: Depends on what the problem is. Start by double checking you have the JDBC connecter added to your project as an external library. Once you are sure it's there, double check you have the username, password, and URL correct (it has a VERY specific format). This encompasses 99% of the issues involving the connection.

Q: What is the best way to check my program?

A: Use the MySQL workbench (or whatever you used for part 2) and look at the tables as they update. If I add a customer to my DB using java, I should see that customer appear in my DB if my connection is set up and everything works. Same goes for pizza, orders, etc.