Project Part 3:
Java Application

For this part of the project, you will implement a Java application that interacts with your Part 2 database. You will implement the functionality needed 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. 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 will complete the project by making changes to 2 files: `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).

`Menu.java` is the main entry point for the application. It is setup to use the methods in `DBNinja` to implement the operations described below. You should NOT change `Menu.java`!

Program Requirements:

1. **Add a new order to the database:** You must be able to add a new order with pizzas to the database. 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. You can assume that a topping can only be added to a pizza 1 time and all the Pizzas in an Order must have the same date/time stamp as the Order.
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.
3. **Enter a new customer:** The program must be able to add the information for a new customer and store it in the database. `Menu.java` will prompt for all the necessary information **(phone numbers will 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 oldest to most recent. There are `DBNinja` methods to support these options.
    a. all open orders
    b. all completed orders
    c. all open orders
    d. all orders (open and completed) on a specific date
    e. the last order entered
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 allows the user to select a specific order for viewing its details. The details include the full order type information, the pizza information (including pizza toppings and discounts), and the order discounts.
6. **Mark an order as completed:** Thera are 3 possible "completion points"
    a. Order preparation is complete
       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."
    b. Order Delivered
       An Delivery Order may be complete, but not yet delivered orders that can be marked as delivered.
    c. Order Picked up
       An Pickup Order may be complete, but not yet picked up by the customer.
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. The format of the reports is shown elsewhere.
   (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 have and make sure each connection is properly closed.
11. The autograder will **NOT** use your working database. Instead, 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!!***
12. Your code needs to be secure, so any time you are adding any sort of parameter to a query that is a `String`, you should 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 user `String` variable, then you can use a regular `Statement` instead.

13. 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. There are descriptions of the basic functionality for all the methods in `DBNinja`, use them to guide your implementation.
14. Since you have a fully functional main package, use it test your `DBNinja` implementation. The starter code will compile and run as is (at least enough to see the menu and exit the application), but it won't do anything until you start implementing `DBNinja`.
15. 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 complete and should not need modification. The files you will be editing are `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 It is 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 static class `DBNinja` is where all the work will be done, the code written in the `Menu` class handles the I/O 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 other class lecture material 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 in MySQL Workbench BEFORE implementing them in code…it will save you a lot of debugging time!

If the code in `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, the pizza prices can be rolled up to calculate the order price and costs and any order discounts are then applied to the entire order. When multiple discounts are applied, make sure you apply the $ discounts before applying the % discounts. 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.).

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**.

When you upload the code to Gradescope, the autograder runs automatically. There are a large number of test cases to run, so it will take be a few minutes before you get your feedback The autograder will tell you how many submissions you have remaining after each run. So, **test your program carefully before submitting!** Use the autograder test input to debug your code before resubmitting it. 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 unless otherwise specified by the instructor.

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:
Do not wait until the last day to ask questions or get started! This is especially important since you can only get autograder results once per day.

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 a 0. This applies to your SQL as well as your Java.**

Note that your code will be tested on an Ubuntu (ie case sensitive) server using a new database created from your SQL scripts. The autograder shows you the exact input it uses for the 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()
	}
}
```

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 (`DBNinja.java and DBConnector.java`)

A: No. You cannot delete or change the method definition of anything in these 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 guide your implementation.

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

A: Maybe. You can implement part 3 using a database that successfully implements part 2. If your part 2 database has issues, then you will need to fix them for part 3.

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 no points. As noted above, you can submit as often as you like…but the autograder will restrict when and how much feedback it provides.

Q: How am I supposed to format the reports?

A: The only output you need to format is for the reports. The reports should be formatted as shown here:

```
(a) ToppingPopularity
Topping             Topping Count
-------             -------------
Pepperoni           17
…<data omitted for conciseness>
Jalapenos           0


(b) ProfitByPizza
Pizza Size          Pizza Crust         Profit              Last Order Date
----------          -----------         ------              ---------------
Small               Original            5.53                4/2024
…<data omitted for conciseness>
Large               Original            69.48               3/2024
```

(c) ProfitByOrderType:

Customer Type	Order Month	Total Order Price	Total Order Cost	Profit
dinein	3/2024	19.75	3.68	16.07
dinein	4/2024	19.78	4.63	15.15
…<data omitted for conciseness>				
delivery	7/2024	11.28	3.70	7.58
	Grand Total	325.76	82.15	243.61

Q: How does partial credit work?

A: That's on a case-by-case basis. But in general, there is no partial credit. There are many test cases and they are very fine grained and provide ample opportunity for 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!

Q: Do we have to use GitHub?

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

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

A: You can always get a connection setup. Follow the instructions and if you have problems, ask for help! If you have issues, 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 your database tool to 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. As said earlier, test your SQL before you use it in your Java.