Lab 2 (Part 1)
CS160L-1001-1002 SUMMER 2023
Genesis Anne Villar
30 May 2023

CS160Lab 2 Part 1- UML Diagram

To Java Coffee Co.

First, I want to thank your company for tasking me to innovate your ordering system for such a prestigious company. I've read through your problem description and laid a foundation for your new ordering system. Please take a look at the provided UML I have sent over. I've broken it down into a few distinct classes as it would appear that these are the classes that fill out the requirements you have asked us to complete.

The first class I would like to briefly touch on is the *menu* class. This is mainly for the customers as it just lists all the add-ons and the coffees your company offers and prints it.

The *coffee* class is responsible for the creation of the coffee. The fields of *coffee* include the type of coffee a customer orders, any add-ons they so wish to add, and the price of the coffee. The type of coffee can range from 1 - 30 – as to include the 30 different types of coffees you stated in your inquiry. Add-ons can be continuously added in the class as to create any flavorful customizations a customer wants. We will be able to set the price as well for the particular coffee order.

The *order* class is responsible for the coffee order system. The field of *order* includes the customer, the order digits (or identification numbers), the coffee the customer orders, the quantity, and the time and date of the order. An order is generated with the customer's name (which will be recorded), a given order number, and time stamped for any record-keeping your company wishes to do. By retrieving the coffee data that the customer creates, we can calculate the overall sale of this customer's particular order. We will be also able to add additional coffees the customer desires and add them to their order. If any errors occur during the ordering process, they will be saved onto the program and can be reviewed/printed later on.

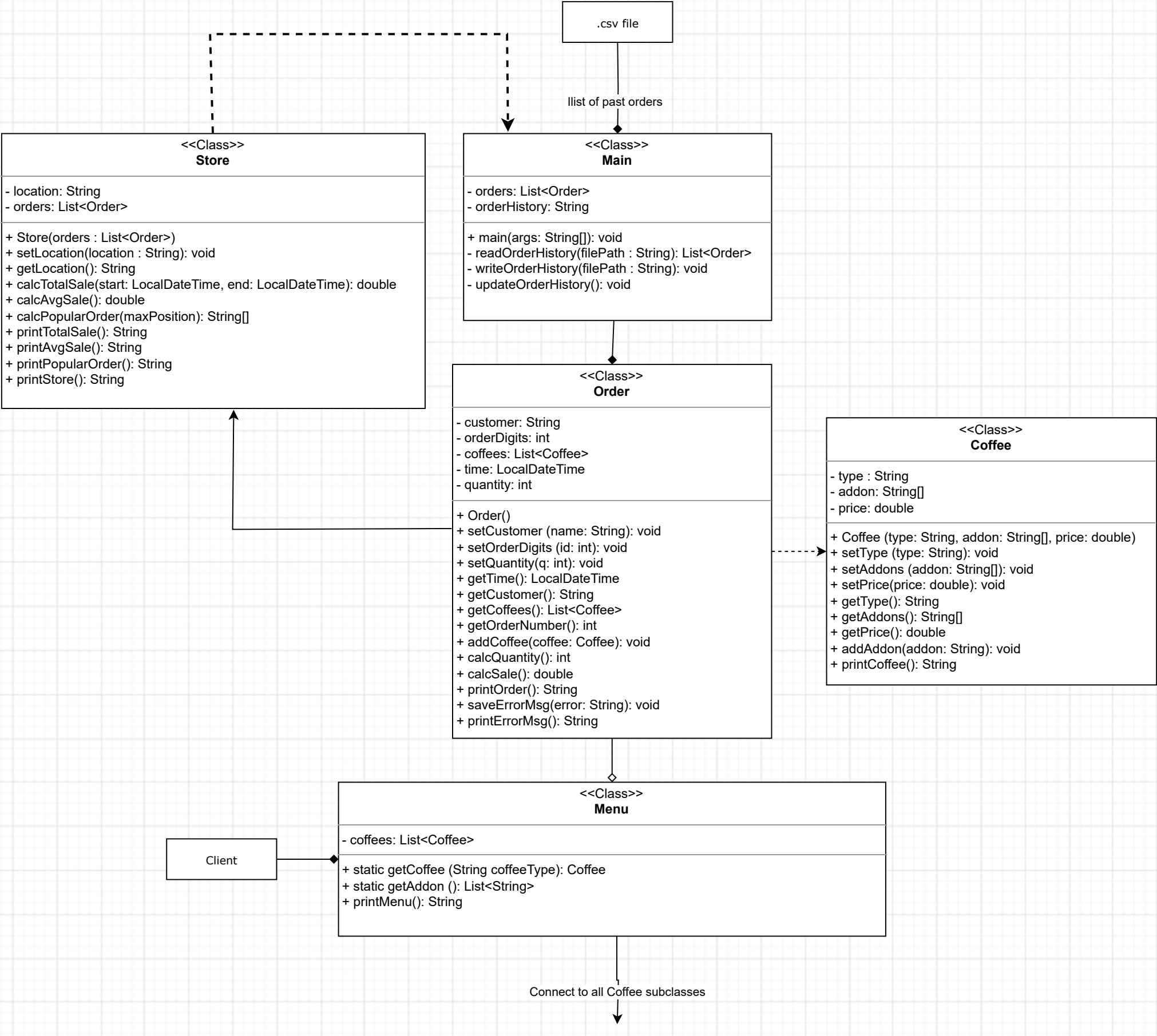
Another class proposed will be the *store* class as this will ultimately track, display, and store information about coffee orders for any given location as you had outlined in the problem statement. The fields of *store* include all the coffee orders created from *order* class, as well as the location name of the data gathered. Retrieving all the information gathered by the coffee creation and the coffee ordering system, we will be able to calculate some store statistics such as the total sale of the store location, calculate the average sale based on the *order* class, and we will be able to determine the most popular coffees created. These business results can all be printed and displayed.

Finally, as we had discussed before, all orders are documented and to be saved onto a .csv file. Assuming all orders are saved and timestamped, the .csv file of each particular store will have all the order history data you so desire. This is accessed by the *main* class. The *main* class stores all the particular store's order history. Additionally, your company may update and write on the order history if you so feel inclined to.

Once again, I thank you for your time, and I am looking forward to continuing our partnership. If there are any questions or issues you may have, please feel free to contact me.

Thank you,

Genesis Anne Villar


.csv file
llist of past orders
<<Class>>
Store
- location: String
- orders: List<Order>
+ Store(orders : List<Order>)
+ setLocation(location : String): void
+ getLocation(): String
+ calcTotalSale(start: LocalDateTime, end: LocalDateTime): double
+ calcAvgSale(): double
+ calcPopularOrder(maxPosition): String[]
+ printTotalSale(): String
+ printAvgSale(): String
+ printPopularOrder(): String
+ printStore(): String
<<Class>>
Main
- orders: List<Order>
- orderHistory: String
+ main(args: String[]): void
- readOrderHistory(filePath : String): List<Order>
- writeOrderHistory(filePath : String): void
- updateOrderHistory(): void
<<Class>>
Order
- customer: String
- orderDigits: int
- coffees: List<Coffee>
- time: LocalDateTime
- quantity: int
+ Order()
+ setCustomer (name: String): void
+ setOrderDigits (id: int): void
+ setQuantity(q: int): void
+ getTime(): LocalDateTime
+ getCustomer(): String
+ getCoffees(): List<Coffee>
+ getOrderNumber(): int
+ addCoffee(coffee: Coffee): void
+ calcQuantity(): int
+ calcSale(): double
+ printOrder(): String
+ saveErrorMsg(error: String): void
+ printErrorMsg(): String
<<Class>>
Coffee
- type : String
- addon: String[]
- price: double
+ Coffee (type: String, addon: String[], price: double)
+ setType (type: String): void
+ setAddons (addon: String[]): void
+ setPrice(price: double): void
+ getType(): String
+ getAddons(): String[]
+ getPrice(): double
+ addAddon(addon: String): void
+ printCoffee(): String
Client
<<Class>>
Menu
- coffees: List<Coffee>
+ static getCoffee (String coffeeType): Coffee
+ static getAddon (): List<String>
+ printMenu(): String
Connect to all Coffee subclasses