Food Ordering System

Scenario

The Food Ordering System (FOS) is a small-scale application that allows any restaurant to manage operations more easily. The system supports multiple roles including the worker, customer and driver. The customer can place order and view the menu. The worker and driver manage orders and handle deliveries.

Design Paradigm: User

Worker:

- Check if the food is on the menu (manage requests)
- Process orders of the customer
- Refund order if it is wrong
- Keep track of each category food is ordered
- View the menu
- Charge the customer

Customer:

- Search food by its type of food
- Search food by its pricing
- Search food by its name
- Order food
- View the menu
- Pay for food
- Choose type of order (deliver or in restaurant)

Doordash driver:

- Accept or refuse orders
- Receive orders
- Charge the customer

Expected Output

The worker can also view the menu. The food can be searched by its pricing, its category of food and its name. The system will also allow them to keep track of how many things they sell from each category. The worker can search for the food on the menu whether it is there, they can also charge

the customer, they can also refund the customer in case of a mistake. It allows them to process orders from the customer. They check if the food is on the menu to manage their requests. Customers can also search for the food based on type, price and name. They are also able to view the menu. The customer will be able to order too. Lastly, the drivers can manage the orders they get and they can view the order they receive. They can accept or refuse the orders if the customer lives too far. This is a food ordering and delivery platform with roles for customers, workers and drivers.

Specification

Hierarchies:

To specify the program, there would be a User class and that class is the superclass or the Parent class. This class will allow us to put in all common data fields and common methods for all the Child class to be able to implement these methods without rewriting the whole thing every time. The Child classes of User class would be Worker, Customer and Driver since these classes will inherit from the User class. There will also be the Food class, but this class is its own independent class just like Order class. It won't be related to the User class.

Interface:

For the interface, we will have a Chargeable interface, so that this allows the workers and driver to charge the customers so that the customers can pay for their orders. The reason why there is an interface Chargeable is because multiple classes will implement the charge method, but not all classes. It would be long to rewrite it for each class that implements it.

Runtime-polymorphism:

There are a couple methods that will use runtime polymorphism. For example, viewing the menu or searching for specific food or displaying their info too. The charge method will also apply runtime polymorphism, because there are two classes that will use that method and so we'll need to specify which one specifically.

TextIO:

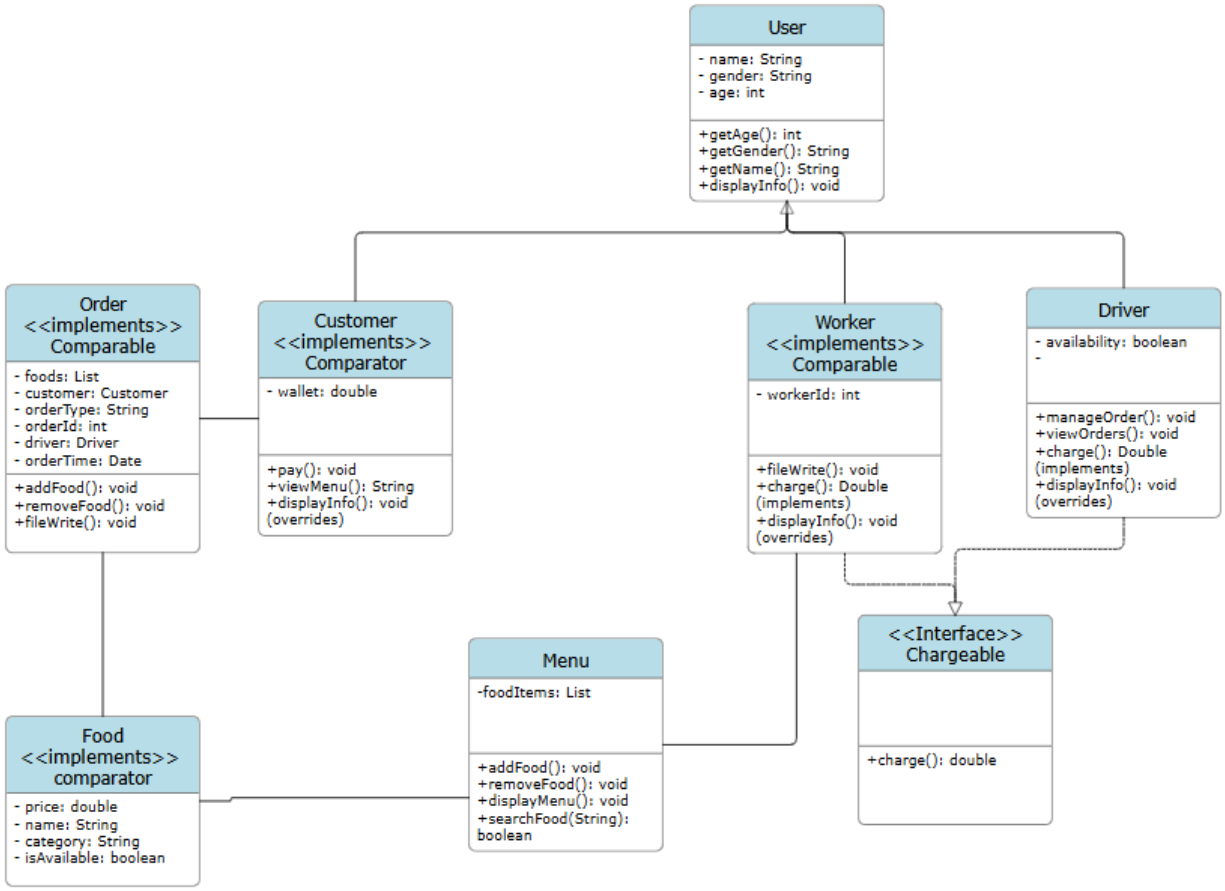
For the text IO, it would be in the worker class, because they would be able to see the past orders of the employees. In the text IO file, it will add the name of the customers and the price of their order and the number of their order. This will allow the workers to find old orders and possibly refund orders using the text IO to check old orders to verify. It could also keep track of sales. It could also be inside order class, since we could track the order id and check what would be inside the order unlike worker class where it shows only total, order id and the customer who ordered.

Comparator and comparable:

The customer class would implement both comparator and comparable, because keeping track of customer data is more important. The customers wouldn't care about the data, but the restaurant

and the admins would. It would be easier for them to keep track of data. The Food and Order class will definitely implement comparator and comparable to sort the food and the number of order. It will be easier to organize the data. Workers could implement a comparator since we might also need information on the workers to keep their data and make it easier to sort and organize the data.

Class diagram:



Deliverable 2 implementation:

The parts that will be implemented in deliverable 2 will be the class Food, Workers, Drivers, Customers and User. Chargeable will also be implemented. All constructors, equals, hashcodes and toString() methods will be implemented already. All getters and setters will be implemented. All data fields will also be defined already in the classes. Most Junit testing too will be already added the Testing file. Most methods will only have the headers implemented. So the methods won't have the body yet. They will be written //TODO inside.