Project Summary

Group 25: Xi Chen, Zhipeng Liu, Weijun Sun, Chenyu Yan, Weijia Zhao

We implement **3** features in group project A5:

1. Robustness
2. Time Sensitive Coupons
3. Solar-powered Drone

Run Project

Language: Java

Framework: Spring Boot JPA

Database: MySQL

To run the program:

- `Right click on pom.xml - Add as Maven Project`

Instruction to run application in docker:

Step 1: stop any Server which is using port 3306

Step2: Prepare Docker images of DroneDelivery

- `docker build -t gatech/dronedelivery -f Dockerfile ./application`

Step3: Run the application in docker

- `docker-compose -p gatech -f docker-compose.yml up --build -d && docker attach app`

Check information saved in database:

- `Create a connection using MySQL workbench (Port: 3306, Username: root, password: omscs2023)`

Rebuild program in docker:

- `Stop and delete all containers and images, restart docker`
- `Build program in docker by following the instruction above`

To run app.jar:

- `Start MySQL server`
- `cd <app.jar directory>`
- `java -jar app.jar`
- `Create a local connection MySQL workbench (Port: 3306, Username: root, password: omscs2023) to check information saved in database`

Modifications and Improvement

Time and Distance System

1. We require the user to input the virtual time at the end of each command, in the format of "command,[parameters],time" and the format of time is "yyyy-MM-dd-hh-mm". Eg: "display_stores,2024-03-01-02-30"

   - Drone charge speed and fly speed are expressed in the unit of minutes

2. To avoid potential confusion in the virtual time system and mimic the reality, we require that users' input time follows the chronological order, i.e. the associated time of a later command cannot must also be later than the associated time of an earlier command. Specifically, we check the timestamp of each command before the actual execution and any timestamp violating the chronological order will cause an error

   - We assume that the machine delay (not include the actual time needed to fulfill the requirement such as delivery) to store/process the command is 1 second, so no two consecutive commands can have exactly the same timestamp. (This is very natural as the system only supports a a single type of user and the user is not able to input two lines of commands at exactly the same time)

3. Location system in the project is implemented in a logical fashion (coordinate system). We keep track of the following locations

   - Store location: input parameters associated with command "make_store"

   - Order destination: input parameters associated with command "start_order"

- Drone location: current location of the drone and the store location of the drone. Auto-computed variables used to compute time of each delivery.

- The distance between two location is calculated by [(x1-x2)^2+(y1-y2)^2]^0.5

4. To be consistent with the A3, we force all numerical attributes/variables to be integers, including the calculated distance (involves square root)

Robustness

1. Add extra validations to ensure the validity of user input
2. Shift program data from in memory storage to MySQL database, allowing

   - Data recovery without loss in the event of a system crash

   - Continue execution without the need of repeatly input the same piece of information in the event of a system crash

3. "Atom" execution for command involving multiple updates of system data: system will only write the update to the database if the entire block of code related to a command is executed succesfully and any errors in the process will lead to fully rollback of the partial updates

Time Sensitive Coupon

In the previous phase, we proposed a method for distributing coupons but found it to be cumbersome and potentially confusing for users due to the number of parameters required for manual input and their interwoven relationships (Eg: we required both the “ex-ante” probability of getting a coupon in each round and the “ex-post” frequency of getting a coupon in HW4 submission and these two parameters are obviously closely related.)

1. We create a method called "make_coupon" which allows the creation and storage of a coupon in the database using three parameters: couponID, expiration date and discount value. In addition to these attributes, the Coupon class also includes a coupon state (valid or expired) and a list of customers who have been assigned the coupon.

2. The "distribute_coupon" method which requires three parameters: coupon ID, base frequency, and high frequency. This method divides customers into two groups based on their rating values and assigning a different frequency to each group. Customers in the top 10 percentile and the bottom 10 percentile receive the high frequency to reward high-value customers and encourage new and inactive customers to continue making purchases. The rest of the customers receive the base frequency. The method randomly assigns coupons to customers based on their frequency assignment.
3. At the time of "start_order", the system will first check if the customer has any coupons assigned and assign the first coupon ID to the order if there is one.
4. At the time of "purchase_order", the system will check if the order was assigned a coupon and deducted the discount from the total order cost if the coupon is still valid.
5. If the order has a coupon attached and it is not delivery before the coupon's expiration, the customer would be refunded 10% of the order cost.

Solar Drone

1. Charge speed: We distinguish the charge speed at daytime/night and daytime is defined as [7:00AM,7:00PM)
2. As users cannot input two commands with exactly the same time, we expand the "purchase_order" command, allowing the input of list of orders from the same store to enable the drone deliver multiple order in a single trip. "purchase_order,storename,order1,order2,...orderN,timestamp". The system will

   - Check the validity of each order: each order must be from the store given by the customer

   - Group the orders according to the associated drones (as we allow the possibility that the orders submitted are attached to different drones)

   - Each drone flys out and get the orders delivered, update the associated parameters along the way. Regardless of the number of orders delivered in a single round (starts when drone leaves the store and ends when the drone comes back), we increment the pilot experience by 1 and decrement the remaining trips before maintanence by 1

3. To get the order delivered, we assume

   - If the drone has multiple orders to deliver in a single trip, the we prioritize orders with coupons and orders for which the attached coupons are close to expire

- If the distance to the next destination is far away that the amount of fuel remained is not enough, the drone will travel until all fuel used up, then stop and get charged with sunlight and then resume again (with the possibility of multiple rounds of refueling if the distance is extremely far away)

- We track the available time of the drone along the way. If a new order is purchased when the drone is still out, the drone will first get all the previous delivery duties fulfilled and then fly back to the store to pick up the new orders.