Supermarket Program

Arab Academy for Science, Technology & Maritime Transport

College of Computing and Information Technology

Course: Object-Oriented Programming
Sheet: Composition, Inheritance, ArrayList, Polymorphism

Project: Supermarket Order System

Problem Description

Design a program that simulates an order from a Supermarket.

The supermarket has different types of items. Each item has a name and price; however, different types of items may have specific attributes. For example:

- **Fruits and Vegetables** have a `weight` attribute, and the price is considered as per the specified weight in KG
- **Dairy products** have an `expiryDate` and `brandName`

To create any item in the system, its name and price must be specified.

Customer and Order Management

To make an order in the system, the customer's data must first be entered:

- `name` , `address` (consists of `city` , `street` , `building` , and `phoneNumber`)

The customer then can start adding products to the order.

Order Details

An order has:

- `Auto-generated ID` , `Creation date` , `List of products` .
- Method that calculates the total price of the order (including delivery fee)

Note: If an order exceeds 500 EGP, then the delivery is free.

Add any necessary methods to the order class that would be needed (e.g., cancel order, etc.) and override the `toString()` method to return all data in the form of an invoice containing all specific details of the order.

Relationships

- The customer has a list of all of their created orders
- An order must also specify to which customer it belongs

Implementation Requirements

The main method should act as a menu of actions for the users that allows interaction with the system.

Note: Start by designing the Class Diagram for the system then translate it into code.

Extra Self-Study Points

1. Create an order status variable that has one of the following values (submitted, prepared, delivered, cancelled) using **Enumeration**
2. Update the order class to keep track of the required quantity of each added item instead of adding the item multiple times using **HashMap**
3. Update the stock class to use a **HashMap** and keep track of the available quantities for each item
4. Update the program to only add items to an order as long as a stock exists for that item.