Description:

Supermarket management system must allow *store managers* to manage the supermarket’s inventory, sales, cashiers, customers, suppliers, and reporting.

Classes:

Manager   
 Product   
 Sale   
 Cashier   
 Customer   
 Supplier   
 Shipment 

Functional Requirements:

The Supermarket Management System provides different types of services where the store manager is the main user; We assume that the store does not have self-checkout.

The functions that are available to the manager include:

* Seeing if an item is out of stock
* Get a list of items that are low (20% by default, or defined in constructor) or out of stock
* Add, remove, and modify products
* Seeing how much of an item was sold in a day
* Seeing how many sales happened in a day (per checkout basis)
* Get average checkout price
* Seeing the most sold items in a day (or month, or year)
* Receive alerts about upcoming shipments
* Get a supplier, date, and list of inventory
* Track purchases for customers
* Add, Remove, and Update information about cashiers
* Add, Remove and Update information about suppliers
* Add, Remove, and update information about upcoming shipments from suppliers
* Add, Remove, and Update information about customers

Classes

**Manager/Main Class**

int currentMonth //the month (mm format (01 is january))

int currentDay //the day (dd format 01 is first of the month)

int currentYear //the year (yyyy format)

String latestLowStockReport //most recently generated lowStockReport

ArrayList<Cashier> cashiers

ArrayList<Supplier> suppliers

HashMap <productName: String, product: Product> inventory //because product names should be unique

ArrayList <Integer> inventoryPrice //indices of products in inventory arranged in ascending order by price, used for ease of access in displayInventory() method

ArrayList <Integer> inventoryStock //indices of products in inventory arranged in ascending order by stock percentage, used for ease of access in displayInventory() method

ArrayList<Customer> customers

ArrayList<Sale> sales //it’d be good to make sure sales are added in reverse chronological order to save time on searches

-----------------------------------------------------------------------------------------------------------------

Manager(currentMonth: int, currentDay: int, currentYear: int) //variables to initialize on program startup, prompt user for the current date

void incrementDay() //Updates currentMonth, currentDay, and currentYear to represent the next day as per the calendar.

void generateLowStockReport () //creates a csv (file) containing the names of all products that are low supply //also updates latestLowStockReport variable //prints success or failure of operation

String[][] getUpcomingShipments(timeframe: String) //returns a table where each column is the supplier, and the rows after are the shipments with their information //timeframe will either be “day” (for shipments of the day), “month” (for shipments in the current month), or “year” (for shipments across the year).

String[][] getUpcomingShipments() //without a parameter this function returns all upcoming shipments in the same format as above

String receiveShipment(supplier: Supplier, id: int) //removes the shipment of matching id from the supplier’s upcomingShipments and updates inventory accordingly. //returns a String informing whether the operation was successful or not and any errors (such as overcapacity on a product or shipment id number not existing).

Product searchInventory(productName: String) //searches inventory and returns Product object that shares a name with productName argument

void displayInventory (criteria: int, toFile: boolean) //prints the inventory arranged in different ways depending on the criteria argument, 0 for chronological, 1 for alphabetical, 2 for price descending, 3 for price ascending, 4 for percentage stock descending, and 5 for percentage stock ascending. //if toFile is true, will instead create a file with the method output rather than print to screen. A message will be printed to inform the user when the file is successfully created. This variation creates files with a naming schema differing by date and will overwrite a file if it shares an exact date with the one being created.

String mostSoldItem(timeframe: String) //returns the name of the most sold item within the time period passed, either “day”, “month”, or “year”

int numSold(productName: Product, timeframe: String) //returns the number of times the specified item has been sold within the timeframe (either “day”, “month”, or “year”)

int numSales(timeframe: String) //returns the number of sales that occurred within specified timeframe (either “day”, “month”, or “year”)

double averageSalePrice(timeframe: String) //gets the average total cost of all sales within the specified timeframe (either “day”, “month”, or “year”)

void addProduct(productName: String, cost: double, currentStock: int, maxStock: int, lowPercentage: int) //creates a new product object and adds it to Hashmap inventory (product names must be unique, will update product instead by behavior of hashmap, somehow indicate the difference to user

void removeProduct(productName: String) //removes product from inventory

//basic cashier management methods

void addCashier(firstName: String, lastName: String, id: int, job: String, salary: int) //creates a new cashier and adds to corresponding arrayList

Void removeCashier(id: int) //removes cashier by id number

Cashier searchCashierByName (String first, String last) //returns cashier that matches first and last name

Cashier[] searchCashierByName (String first) //returns a list of all cashiers with matching first name

void displayCashiers() //prints out a list of all cashiers and their information

//basic customer management methods

void addCustomer(firstName: String, lastName: String, rewardsNumber: int, phoneNumber: String) //creates a new customer and adds it to corresponding arrayList //purchasesMade attribute is set to 0 by default

void removeCustomer(rewardsNumber: int) //removes customer based on rewards number (should be unique for each customer)

Customer searchCustomerByName (String first, String last) //returns customer that matches first and last name

 void displayCustomers() //prints out a list of all customers and their information

//basic sale management methods

void addSale(dayOfSale: int, monthOfSale: int, yearOfSale: int, managingSale: Cashier, makingSale: Customer) //creates new sale object and adds it to the front (or 0th index) of Sales arrayList

void addSale(dayOfSale: int, monthOfSale: int, yearOfSale: int, managingSale: Cashier) //creates new sale object and adds it to the front (or 0th index) of Sales arrayList, this one does not require a customer to be passed

void removeSale(index: int) //removes sale object by its index in arrayList

void displaySales () //displays all sales with their information

**Supplier**

String name //name of supplier

String[] products //names of products delivered by the supplier

HashMap<id: Integer, shipment: Shipment> upcomingShipments

----------------------------------------------------------------------------------------------------------------

Supplier (name, products) //constructor

String[] getLowProducts (masterReport: String) //parses the total low stock report from the generateLowStockReport() method in manager class, returns an array of all product names that are supplied by this supplier (defined by String[] products attribute)

void addShipment(id: int, arrivalMonth: int, arrivalDay: int, arrivalYear: int) //creates a new shipment and adds it to upcomingShipments

void updateShipment(id: int, arrivalMonth: int, arrivalDay: int, arrivalYear: int) //updates shipment with corresponding id number

void removeShipment(id: int) //removes a shipment by its id, does nothing if it doesn’t exist

**Product**

String name

double cost

int currentStock //current amount of this item on hand

int maxStock //total amount of this item the store can hold

int lowPercentage //threshold for defining the product as “low” as percentage

--------------------------------------------------------------------------------------------------

Product (name: String, cost: int, currentStock: int, maxStock: int) //default low percentage is 20%

Product (name: String, cost: int, currentStock: int, maxStock: int, lowPercentage: int)

int getStockPercentage () //returns the current stock level as a percentage //floors percentage if not whole

double increasePrice (percentage: double) //increases the price based on percentage passed, returns new price

double decreasePrice (percentage: double) //decreases the price based on percentage passed, returns new price

**Shipment**

int idNumber //an id number for the shipment

int arrivalMonth //month of arrival

Int arrivalDay //day of arrival

Int arrivalYear //year of arrival

HashMap<productName: String, product: Product> items //name of products coming in and how many

-------------------------------------------------------------------------------------------------------------

Shipment(id: int, arrivalMonth: int, arrivalDay: int, arrivalYear: int) //productList is empty

String arrivalDateToStr() //returns the date of arrival as mm/dd/yyyy for reporting purposes

void addProduct(name: String, cost: double, currentStock: int, maxStock: int, lowPercentage: int) //adds a new product to product list, note that product names must be unique, otherwise the product will be updated instead. //also the product name and the name it is mapped to in the hash map must match as well. //if the product does not exist in inventory yet, ask the user to define a maxStock and lowPercentage for the product, otherwise use the values of the product that is already there.

void removeProduct(productName: String) //removes item from list

**Cashier**

String firstName

String lastName

int id

int salary

---------------------------------------------------------------------------------------------------------------

Cashier(id: int, firstName: String, lastName: String, job: String, salary: int)

void increaseSalary (percent: double) //increases the salary of the cashier by a percentage

**Customer**

String firstName

String lastName

int customerId

String phoneNumber

int purchasesMade

-----------------------------------------------------------------------------------------------------------------

Customer(firstName: String, lastName: String, customerId: int, String: phoneNumber)

void incrementPurchase() //increments purchasesMade variable by +1

**Sale**

int dayOfSale

int monthOfSale

int yearOfSale

Cashier managingSale

ArrayList <Product> productsSold

Customer makingSale

-----------------------------------------------------------------------------------------------------------------

Sale(managingSale: Cashier, productsSold: ArrayList <Product>) //Constructor is if the Customer is not in the system

Sale(managingSale: Cashier, productsSold: ArrayList <Product>, makingSale: Customer) //Constructor if customer is in the system

int numberOfProductsSold() //Returns how many items were involved in the transaction

double totalOfSale() //Returns the total of the transaction

void addItem(item: Product) //adds passed product to productsSold arraylist

A computer screen shot of a computer

AI-generated content may be incorrect.