**Members:**

**Opiniano, Ezkylle**

**San Pedro, Adrian Elijah**

**Manio, Marco Ruiz**

**Week 2: Core Functionality (Java Code)**

**1. DeliveryAgent Class**

This class will represent a delivery agent with attributes like name, contact number, vehicle type, and availability.

/\*\*

 \* The DeliveryAgent class represents a delivery agent,

 \* encapsulating key details such as name, contact number,

 \* vehicle type, and availability.

 \*

 \* Attributes:

 \* - name: The full name of the delivery agent.

 \* - contactNumber: A string representing the agent's contact number.

 \*   This could be expanded to include validation for format in future improvements.

 \* - vehicleType: Describes the type of vehicle the agent uses

 \*   (e.g., bike, motorcycle, car, van, truck).

 \* - availability: A boolean indicating whether the agent is currently available for deliveries.

 \*

 \* Methods:

 \* - Getters and setters are used to ensure encapsulation, allowing for controlled

 \*   access and modification of agent details.

 \* - toCSV: Converts the delivery agent's information to CSV format for easy storage.

 \* - fromCSV: A static method that allows for the creation of a DeliveryAgent object

 \*   from a CSV line, facilitating CSV imports.

 \*/

public class DeliveryAgent

{

    private String name;

    private String contactNumber;

    private String vehicleType;

    private boolean availability;

    // Constructor

    public DeliveryAgent(String name, String contactNumber, String vehicleType, boolean availability)

    {

        this.name = name;

        this.contactNumber = contactNumber;

        this.vehicleType = vehicleType;

        this.availability = availability;

    }

    // Getters and setters for encapsulation

    public String getName()

    {

        return name;

    }

    public void setName(String name)

    {

        this.name = name;

    }

    public String getContactNumber()

    {

        return contactNumber;

    }

    public void setContactNumber(String contactNumber)

    {

        this.contactNumber = contactNumber;

    }

    public String getVehicleType()

    {

        return vehicleType;

    }

    public void setVehicleType(String vehicleType)

    {

        this.vehicleType = vehicleType;

    }

    public boolean isAvailable()

    {

        return availability;

    }

    public void setAvailability(boolean availability)

    {

        this.availability = availability;

    }

    // Method to convert to CSV format

    public String toCSV()

    {

        return name + "," + contactNumber + "," + vehicleType + "," + availability;

    }

    // Static method to create an agent from CSV

    public static DeliveryAgent fromCSV(String csvLine)

    {

        String[] fields = csvLine.split(",");

        return new DeliveryAgent(fields[0], fields[1], fields[2], Boolean.parseBoolean(fields[3]));

    }

}

**2. DeliveryAgentManager Class**

This class will manage the collection of delivery agents, allowing for registration, updating, deletion, and saving/loading from a CSV file.

/\*\*

 \* The DeliveryAgentManager class manages a collection of delivery agents,

 \* providing CRUD (Create, Read, Update, Delete) operations.

 \*

 \* It is responsible for:

 \* - Registering new agents

 \* - Updating existing agents based on their index in the list

 \* - Deleting agents by index

 \* - Saving agent data to a CSV file for persistence

 \* - Loading agent data from a CSV file on application startup

 \*

 \* The class is designed with extensibility in mind, allowing additional

 \* operations to be added in the future (e.g., searching agents by name or vehicle type).

 \*/

import java.util.ArrayList;

import java.util.List;

public class DeliveryAgentManager

{

    private List<DeliveryAgent> agents;

    public DeliveryAgentManager()

    {

        this.agents = new ArrayList<>();

    }

    // Register a new delivery agent

    public void addAgent(DeliveryAgent agent)

    {

        agents.add(agent);

        System.out.println("Agent added successfully.");

    }

    // Update an existing agent

    public void updateAgent(int index, DeliveryAgent updatedAgent)

    {

        if (index >= 0 && index < agents.size())

        {

            agents.set(index, updatedAgent);

            System.out.println("Agent updated successfully.");

        }

        else

        {

            System.out.println("Invalid agent index.");

        }

    }

    // Delete an agent

    public void deleteAgent(int index)

    {

        if (index >= 0 && index < agents.size())

        {

            agents.remove(index);

            System.out.println("Agent deleted successfully.");

        }

        else

        {

            System.out.println("Invalid agent index.");

        }

    }

    // Save agents to CSV

    public void saveToCSV(String filePath)

    {

        CSVUtils.writeToCSV(filePath, agents);

        System.out.println("Agents saved to CSV.");

    }

    // Load agents from CSV

    public void loadFromCSV(String filePath)

    {

        agents = CSVUtils.readFromCSV(filePath);

        System.out.println("Agents loaded from CSV.");

    }

    // Search agents by name

    public List<DeliveryAgent> searchByName(String name)

    {

        List<DeliveryAgent> result = new ArrayList<>();

        for (DeliveryAgent agent : agents)

        {

            if (agent.getName().equalsIgnoreCase(name))

            {

                result.add(agent);

            }

        }

        return result;

    }

    // Get the list of agents (for displaying or further operations)

    public List<DeliveryAgent> getAgents()

    {

        return agents;

    }

}

**3. CSVUtils Class**

This class will handle the reading from and writing to CSV files for storing agent information.

/\*\*

 \* The CSVUtils class provides utility methods for reading and writing delivery agents

 \* to and from CSV files. This ensures persistence of agent data between program sessions.

 \*

 \* Key Methods:

 \* - writeToCSV: Writes the list of delivery agents to a specified CSV file, handling potential

 \*   I/O exceptions and ensuring that data is saved properly.

 \* - readFromCSV: Reads delivery agent data from a CSV file, creating DeliveryAgent objects

 \*   from each line and returning a list of agents.

 \*

 \* Exception Handling:

 \* - Both methods include error handling to ensure the program continues to run

 \*   smoothly in case of file I/O errors. Errors are logged to the console, giving

 \*   the user insights into any issues that occur during file operations.

 \*/

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class CSVUtils

{

    // Write agents to a CSV file with headers

    public static void writeToCSV(String filePath, List<DeliveryAgent> agents) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {

            // Write the header first

            writer.write("ID,Name,Contact Number,Vehicle Type,Availability");

            writer.newLine();

            // Write each agent's information

            int id = 1;  // Start ID from 1

            for (DeliveryAgent agent : agents) {

                writer.write(id + "," + agent.toCSV());

                writer.newLine();

                id++;

            }

        } catch (IOException e) {

            System.out.println("Error writing to CSV: " + e.getMessage());

        }

    }

    // Read agents from a CSV file

    public static List<DeliveryAgent> readFromCSV(String filePath)

    {

        List<DeliveryAgent> agents = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath)))

        {

            String line;

            while ((line = reader.readLine()) != null)

            {

                agents.add(DeliveryAgent.fromCSV(line));

            }

        }

        catch (IOException e)

        {

            System.out.println("Error reading from CSV: " + e.getMessage());

        }

        return agents;

    }

}

**4. Main Class**

This class will manage the input/output and act as the entry point of the application.

/\*\*

 \* The Main class serves as the entry point for the Delivery Agent Management System.

 \* It provides a user interface to manage delivery agents through various operations

 \* such as adding, updating, deleting, listing, and searching agents.

 \* The application loads agent data from a CSV file on startup and saves it before exiting.

 \*/

import java.util.List;

import java.util.Scanner;

import java.util.InputMismatchException;

public class Main

{

    public static void main(String[] args)

    {

        Scanner scanner = new Scanner(System.in);

        DeliveryAgentManager manager = new DeliveryAgentManager();

        // Use the correct path to the data folder

        String filePath = "C:\\Users\\Ezkylle Opiniano\\Desktop\\SchoolStuff\\1st Semester\\oop\\finals\\Final OOP Group 7\\DeliveryAgentManagementSystem\\Data\\agents.csv";

        // Load agents from CSV (if the file exists)

        try {

            manager.loadFromCSV(filePath);

        }

        catch (Exception e)

        {

            System.out.println("Failed to load data from CSV. Please check the file path.");

        }

        boolean exit = false;

        while (!exit)

        {

            System.out.println("\nDelivery Agent Management System:");

            System.out.println("1. Add Agent");

            System.out.println("2. Update Agent");

            System.out.println("3. Delete Agent");

            System.out.println("4. List All Agents");

            System.out.println("5. Save Agents to CSV");

            System.out.println("6. Exit");

            int choice = -1;

            // Loop to ensure valid integer input for menu selection

            while (choice == -1)

            {

                System.out.print("Choose an option: ");

                try

                {

                    choice = scanner.nextInt();

                    scanner.nextLine();  // Consume newline after nextInt

                }

                catch (InputMismatchException e)

                {

                    System.out.println("Invalid input. Please enter a number between 1 and 6.");

                    scanner.nextLine();  // Consume the invalid input

                }

            }

            switch (choice)

            {

                case 1:

                    // Add a new agent

                    DeliveryAgent newAgent = inputAgent(scanner);

                    manager.addAgent(newAgent);

                    break;

                case 2:

                    // Update an agent

                    System.out.print("Enter agent index to update: ");

                    int updateIndex = getValidIndex(scanner, manager.getAgents().size());

                    DeliveryAgent updatedAgent = inputAgent(scanner);

                    manager.updateAgent(updateIndex, updatedAgent);

                    break;

                case 3:

                    // Delete an agent

                    System.out.print("Enter agent index to delete: ");

                    int deleteIndex = getValidIndex(scanner, manager.getAgents().size());

                    manager.deleteAgent(deleteIndex);

                    break;

                case 4:

                    // List all agents

                    List<DeliveryAgent> agents = manager.getAgents();

                    if (agents.isEmpty())

                    {

                        System.out.println("No agents available.");

                    }

                    else

                    {

                        System.out.println("ID,Name,Contact Number,Vehicle Type,Availability"); // Header for display

                        for (int i = 0; i < agents.size(); i++)

                        {

                            System.out.println((i + 1) + ": " + agents.get(i).toCSV());  // Start displaying agents from 1

                        }

                    }

                    break;

                case 5:

                    // Save agents to CSV

                    manager.saveToCSV(filePath);

                    break;

                case 6:

                    // Exit the program

                    System.out.println("Exiting program.");

                    manager.saveToCSV(filePath);

                    scanner.close();

                    exit = true;

                    break;

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

        }

    }

    // Method to input agent details from the user

    private static DeliveryAgent inputAgent(Scanner scanner)

    {

        System.out.print("Enter agent name: ");

        String name = scanner.nextLine();

        System.out.print("Enter agent contact number: ");

        String contactNumber = scanner.nextLine();

        System.out.print("Enter vehicle type: ");

        String vehicleType = scanner.nextLine();

        boolean availability;

        while (true)

        {

            System.out.print("Is the agent available? (true/false): ");

            String availabilityInput = scanner.nextLine();

            if (availabilityInput.equalsIgnoreCase("true") || availabilityInput.equalsIgnoreCase("false"))

            {

                availability = Boolean.parseBoolean(availabilityInput);

                break;

            }

            else

            {

                System.out.println("Invalid input. Please enter 'true' or 'false'.");

            }

        }

        return new DeliveryAgent(name, contactNumber, vehicleType, availability);

    }

    // Method to validate and get a valid index for update/delete operations

    private static int getValidIndex(Scanner scanner, int size)

    {

        int index = -1;

        while (index < 0 || index >= size)

        {

            System.out.print("Enter a valid index: ");

            try

            {

                index = scanner.nextInt();

                scanner.nextLine();  // Consume newline after nextInt

                if (index < 0 || index >= size)

                {

                    System.out.println("Index out of range. Please try again.");

                }

            }

            catch (InputMismatchException e)

            {

                System.out.println("Invalid input. Please enter a valid number.");

                scanner.nextLine();  // Consume the invalid input

            }

        }

        return index;

    }

}

Output: