**ID:** 13474078970

**Name:** Umut Uygur

**Section:** sec\_01

**Problem Summary:** The task involved utilizing Java to create a simulation software for freight transportation. This software loads, drops, and delivers items throughout many cities using assigned vehicles after reading mission data from a file.Each mission specifies which packages will be transfer to the correct destination, the loading and delivery sequence, and for dropping packages at mid-city. The application also generates an output file (result.txt) the final status of cargos and vehicles in each city.  
 **Sub-Tasks:**  
  
**Loading Packages from start**: Move the assigned packages which described into mission from the source city to the vehicle.  
**Additional Loading in Middle City**: Load specified packages in the mid city.  
**Dropping Packages in the Middle City**: Drop specific packages from the vehicle mentioned in the mission.  
**Final Delivery**: Bring the remaining packages to their final city.  
**Output Formatting:** Input the final package and vehicle deployment into result.txt in the correct format

* **main();** orchestrates the program by reading missions and executing each mission.
* **readMissions();** reads the input data from files and parses it into mission structures.
* **executeMission();** performs each mission by loading, dropping, and delivering packages.
* **writeResultsToFile();** writes the final city-by-city package and vehicle distribution to result.txt.

**Implementation and Functionality**

**readMissions()**

* **Purpose:** Reads mission information from a file and initializes the relevant city, package, and vehicle data.
* **Inputs:** File path for the mission data file.
* **Outputs:** A list of missions with parsed data.
* **Code;**

**public List<Mission> readMissions(String filePath) {**

**List<Mission> missions = new ArrayList<>();**

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

**String line;**

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

**// Reads line from file and converts it to task**

**Mission mission = parseMission(line);**

**missions.add(mission);**

**}**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**return missions;**

**}**

**executeMission(Mission mission)**

* **Purpose:** Executes loading, dropping, and delivering packages based on the mission.
* **Inputs**: Mission data containing source, middle, and destination cities, package list, and drop instructions.
* **Outputs:** Updates the state of cities after mission completion.
* **Code;**

**public void executeMission(Mission mission) {**

**// Loading from source city**

**mission.getSourceCity().loadPackagesToVehicle(mission.getPackagesToLoad());**

**// Additional loading and dropping off in intermediate city**

**if (mission.getMiddleCity() != null) {**

**mission.getMiddleCity().loadPackagesToVehicle(mission.getMiddlePackages());**

**mission.getMiddleCity().dropPackages(mission.getPackagesToDrop());**

**}**

**//Delivery to destination city**

**mission.getDestinationCity().receivePackages(mission.getPackagesToDeliver());**

**}**

**writeResultsToFile()**

* **Purpose:** Outputs the final package and vehicle state for each city to result.txt.
* **Inputs:** Final city data structures.
* **Outputs:** result.txt file with formatted results.
* **Code;**

**public void writeResultsToFile(List<City> cities) {**

**try (BufferedWriter writer = new BufferedWriter(new FileWriter("result.txt"))) {**

**for (City city : cities) {**

**writer.write(city.getName() + "\n");**

**writer.write("Packages:\n");**

**for (Package pkg : city.getPackages()) {**

**writer.write(pkg.toString() + "\n");**

**}**

**writer.write("Vehicles:\n");**

**for (Vehicle vehicle : city.getVehicles()) {**

**writer.write(vehicle.toString() + "\n");**

**}**

**writer.write("-------------\n");**

**}**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**TESTİNG**

Testing To ensure the application functions as intended, I manually tested it by executing several scenarios. The primary things I looked at were:  
  
**Loading and Unloading:** I verified that the items were loaded in the right order at the city of origin and emptied at each destination in the proper order.  
**Dropping in mid City** : By the mission instructions, I verified that particular parcels may be dropped at the middle city based on their designated indices.  
**Edge Cases:** To ensure the software handled edge cases appropriately, I additionally evaluated scenarios such as trying to dump packages that weren't in the car or having just one package to unload.  
Every test was successful, and the software consistently generated the desired results.

**Trouble Points:** Problem areas included troubleshooting index issues and managing Stack and Queue data structures for package and vehicle handling.  
  
**Most Challenging Part:** The most difficult aspect of implementing the dropPackages() function was keeping the packages in the right order and avoiding index-related problems while dropping packages.  
  
**What I Liked and Learned:** What I Liked and Learned: I found this project to be interesting, especially the part about organizing and troubleshooting the simulation. It enhanced my ability to handle files in Java and manage data structures (Stack, Queue), as well as to write clear, organized code for modular jobs.