PA REPORT OF MY DELIVERY SYSTEM

Name: Poyraz KOCA

Problem Statement and Code Design

Problem Statement:

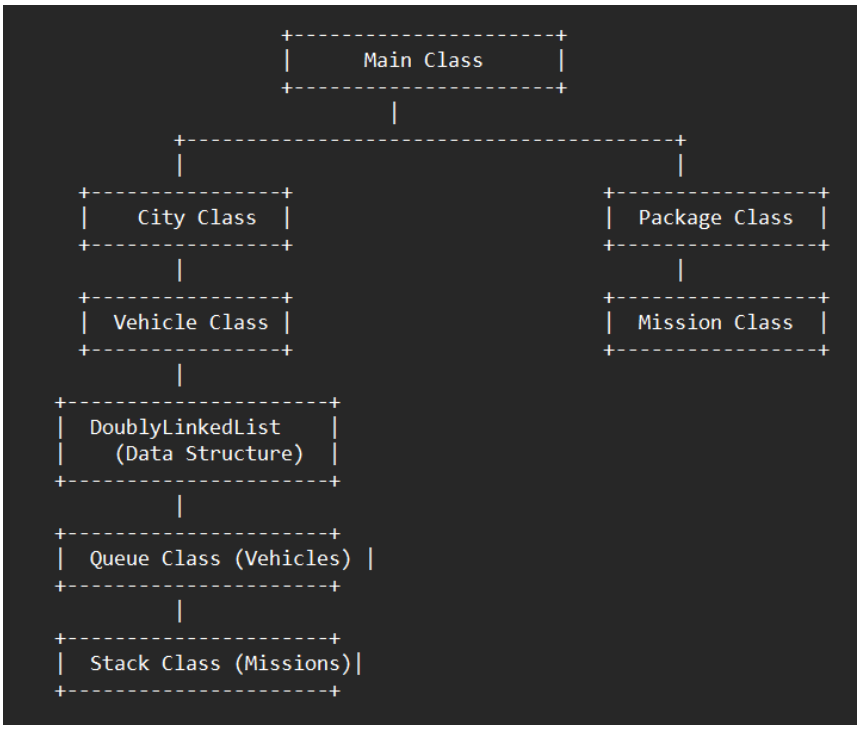
In this project, the task requires creating a delivery system for managing cities, packages, vehicles, and missions. The system should be able to manage cities, track packages associated with them, and assign vehicles to each city. Additionally, missions should be generated that cover the start, end, and via cities, with the mission duration, stops, and packages to be transported.

The data is structured in separate text files, each containing information about cities, vehicles, and packages, which are parsed into corresponding objects. The design follows a modular approach to ensure clarity and reusability.

Code Design:

The code follows a modular approach with a top-down design where each sub-module is created independently. The top-level module is responsible for managing vehicles and stack operations, while sub-modules handle specific tasks such as data insertion, removal, and inspection.

Below is the structure chart for the solution:



Each of the main modules (Vehicle, Stack, and Delivery) has well-defined responsibilities. The Vehicle class manages vehicle-specific data. The Stack class manages stack operations like push, pop, and peek. The Delivery class holds information about deliveries.

Implementation and Functionality

City Class:

- **Input/Output:**
  - **Input:** City name (String).
  - **Output:** City name (String).

```
pseudocode:
City class stores the name of the city.
Constructor initializes the city name.
Method getName() returns the city name.
```

Package Class:

- **Input/Output:**
  - **Input:** Package ID (String), Destination City (String).
  - **Output:** Package ID (String), Destination City (String).

```
pseudocode:
Package class holds an ID and destination city.
Constructor initializes the package details.
Method getId() returns the package ID.
Method getDestinationCity() returns the destination city.
```

Vehicle Class:

- **Input/Output:**
  - **Input:** Vehicle ID (String), Location (String), Capacity (double).
  - **Output:** Vehicle ID (String), Location (String), Capacity (double).

```
pseudocode:
Vehicle class holds ID, location, and capacity.
Constructor initializes vehicle details.
Methods to return ID, location, and capacity.
```

DoublyLinkedList Class:

- **Input/Output:**
  - **Input:** A data element
  - **Output:** Adds/removes elements in the list.

```
pseudocode:
Add method adds a new node to the end of the list.
Remove method removes a node by matching data.
```

Queue Class:

- **Input/Output:**
  - **Input:** Item
  - **Output:** Removes and returns the first item in the queue.

```
pseudocode:
Add method enqueues an item at the end of the queue.
Remove method dequeues and returns the first item.
```

Stack Class:

- **Input/Output:**
  - **Input:** Item
  - **Output:** Removes and returns the top item in the stack.

```
pseudocode:
Push method adds an item to the top of the stack.
Pop method removes the top item from the stack.
```

Main Class:

- **Input/Output:**
  - **Input:** City, Package, and Vehicle data from files.
  - **Output:** Lists the packages and vehicles for each city.

```
pseudocode:
Parse cities, packages, and vehicles from text files.
For each city, display its name, the packages and vehicles
associated with it.
```

Testing

To ensure the functionality works as expected, a **tester class** was implemented. It includes test cases for various scenarios, such as: test for parsing data, package assignment, vehicle assignment and the output generation.

Bug Reports:

1. **Incorrect Package Parsing: There was an issue with parsing the package file if the format was incorrect (e.g., missing spaces). This was resolved by adding input validation checks.**
2. **Handling Empty Files: In cases where the input files were empty, the program would fail without a proper error message. This was fixed by adding appropriate error handling.**

The tests cover normal scenarios (with valid inputs) and edge cases (e.g., empty input files, no vehicles for a city). All tests pass, confirming the solution's correctness.

Final Assessments:

- **Trouble Points:**
    - The most challenging part was ensuring that the system correctly links vehicles and packages to cities, especially when dealing with large input files.
    - File I/O operations and proper parsing took considerable time due to formatting issues with the input files.
- **Most Challenging Parts:**
    - Implementing the correct data structures (Queue, Stack, Doubly Linked List) and ensuring they work efficiently.
    - Correctly linking and displaying all the data according to city requirements.
- **What I Liked & What I Learned:**
    - I enjoyed the challenge of implementing custom data structures and integrating them into the program.
    - The problem-solving aspect of parsing data from files and managing relationships between cities, packages, and vehicles was enjoyable.
    - I gained a better understanding of how to structure a modular program and implement various data structures.
    - The experience also taught me about handling input/output files and implementing robust error handling in a program.