README FILE

Object-Oriented Concepts which have been demonstrated

- **Inheritance**
  All vehicle classes (`Car`, `Truck`, `Bus`, `Airplane`, `CargoShip`) inherit from the common abstract base class `Vehicle`.
  Example: `Car extends Vehicle`, `Bus extends Vehicle`.

- **Polymorphism**
  The program stores all vehicles in a single `List<Vehicle>` inside `FleetManager`. Even though the list is typed as `Vehicle`, at runtime, the correct subclass method (`displayInfo()`, `startJourney()`, etc.) is invoked based on the actual vehicle type.

- **Abstract Classes**
  `Vehicle` is an abstract class. It defines shared properties (ID, model, maxSpeed) and abstract methods (e.g., `displayInfo()`), which each subclass implements in its own way.

- **Interfaces**
  The `FuelConsumable` interface defines the `refuel(double amount)` method. Vehicles that use fuel (Cars, Trucks, Buses, Airplanes, CargoShips without sails) implement this interface, ensuring a consistent contract across different types of vehicles.

. Compilation and Execution

Compile all files

Navigate to the `src` directory and run:

```
javac exceptions/*.java fleet/*.java interfaces/*.java vehicles/*.java Main.java
```

Run the program

```
java Main
```

Test persistence with CSV

    - Use the menu option `7. Save Fleet` to save the current fleet to a file (e.g., `fleet.csv`).
    - Use the menu option `8. Load Fleet` to reload the fleet from that CSV file. This demonstrates persistence and verifies that the vehicles are stored/restored correctly.

Using the CLI

When you run the program, you will see the **Fleet Manager Menu**:

```
=== Fleet Manager Menu ===
1. Add Vehicle
2. Remove Vehicle
3. Start Journey
4. Refuel All
5. Perform Maintenance
6. Generate Report
7. Save Fleet
8. Load Fleet
9. Search by Type
10. Search by ID
11. List Vehicles Needing Maintenance
12. Exit
```

- **Add Vehicle**: Enter type (Car/Truck/Bus/Airplane/CargoShip) and details (ID, model, speed, wheels, etc.).
- **Remove Vehicle**: Enter ID to delete a vehicle from the fleet.
- **Start Journey**: Simulate all vehicles traveling a given distance. Fuel decreases, and maintenance may be required.
- **Refuel All**: Add fuel to all vehicles that support it.
- **Perform Maintenance**: Reset maintenance status for vehicles.
- **Generate Report**: View details of all vehicles and their status.
- **Save Fleet**: Save the fleet to a CSV file.
- **Load Fleet**: Load fleet data from a previously saved CSV file.

- **Search by Type/ID**: Quickly find vehicles.
- **List Maintenance Needs**: See which vehicles require maintenance.

---

4. Demo Walkthrough and Expected Output

Start the program:

```
java Main
```

1. Add a Car:

```
Enter vehicle type (Car/Truck/Bus/Airplane/CargoShip): Car

Enter ID: C001
Enter model: Toyota
Enter max speed (km/h): 120
Enter numWheels: 4
Car added successfully.
```

2. Start a journey:

```
Enter distance (km) for journey: 100

(Output shows each vehicle simulating travel and fuel consumption.)
```

3. Generate a report:

```
Fleet Report:

Car [ID=C001, Model=Toyota, Speed=120.0, Fuel=...]
```

4. Save the fleet:

```
Enter filename to save fleet: fleet.csv

Fleet saved to fleet.csv
```

5. Exit and reload:

```
Enter filename to load fleet: fleet.csv

Fleet loaded from fleet.csv
```

6. Expected outcome: Vehicles persist across program runs, the user can add/search/remove vehicles, simulate journeys, refuel, and maintain the fleet.