**Detailed Explanation of the Car Rental Management System**

This Java program is a **console-based car rental management system** that demonstrates the use of interfaces, enums, and object-oriented principles to manage vehicles like cars, motorcycles, and trucks. Below is a breakdown of how the program works:

**1. Program Overview**

The application allows users to:

* Add vehicles (cars, motorcycles, trucks) with validated inputs.
* Display all stored vehicles with color-coded output.
* Save vehicle data to a file (vehicles.txt).
* Handle invalid inputs gracefully using error-checking mechanisms.

**2. Key Components**

**a) Enums for Input Validation**

* FuelType: Restricts fuel options to PETROL, DIESEL, or ELECTRIC.
* MotorcycleType: Limits motorcycle types to SPORT, CRUISER, or OFF\_ROAD.
* TransmissionType: Ensures transmission is either MANUAL or AUTOMATIC.

**Why?**  
Enums prevent typos (e.g., "desiel") and enforce valid inputs, reducing runtime errors.

**b) Interfaces**

* Vehicle: Defines core methods (getMake(), getModel(), getYear()) for all vehicles.
* **Specialized Interfaces**:
  + CarVehicle: Adds methods for doors and fuel type.
  + MotorVehicle: Adds methods for wheels and motorcycle type.
  + TruckVehicle: Adds methods for cargo capacity and transmission.

**Why?**  
Interfaces ensure that all vehicle classes adhere to a consistent structure while allowing flexibility for unique attributes.

**c) Vehicle Classes**

* Car, Motorcycle, Truck: Each implements the Vehicle interface and a specialized interface.  
  Example:

class Car implements Vehicle, CarVehicle {

*// Implements all required methods from both interfaces*

}

* **Input Validation**: Constructors and setters include checks (e.g., no negative cargo capacity).

**d) Main Program**

* **Color-Coded UI**: Uses ANSI escape codes for colored text (e.g., red for errors, green for success messages).
* **Interactive Menu**:
  + Users choose options to add vehicles, display data, or exit.
  + Inputs are validated using try-catch blocks to handle invalid entries (e.g., entering text where a number is expected).
* **File Saving**: Vehicles are saved to vehicles.txt in CSV format upon exit.

**3. Workflow Example**

1. **Adding a Car**:
   * User selects "Add Car" and enters make (e.g., "Tesla"), model (e.g., "Model 3"), year, doors, and fuel type.
   * The program converts the fuel type input to an enum (e.g., FuelType.ELECTRIC).
   * Data is stored in an ArrayList<Vehicle>.
2. **Displaying Vehicles**:
   * The program iterates through the ArrayList and uses instanceof checks to print vehicle-specific details.  
     Example:

if (v instanceof Car c) {

System.out.println("Doors: " + c.getDoors());

}

1. **Error Handling**:
   * Invalid inputs (e.g., letters instead of numbers) trigger a red error message and prompt the user to retry.
   * Invalid enum values (e.g., "AUTOMATIK") throw an IllegalArgumentException.
2. **File Operations**:
   * On exit, vehicles are saved to vehicles.txt using FileWriter.  
     Example output in the file:

Tesla,Model 3,2025

Harley-Davidson,Street Glide,2023

**4. Unique Features**

* **Colorful Console Output**:  
  Uses ANSI codes like \u001B[31m (red) and \u001B[32m (green) to improve readability.
* **Enum Validation**:  
  Forces users to pick valid options (e.g., ELECTRIC instead of free-text inputs).
* **Polymorphism**:  
  All vehicles are stored in a single ArrayList<Vehicle>, simplifying data management.
* **File Persistence**:  
  Data persists between program runs via vehicles.txt.

**5. Real-World Applications**

* **Car Rental Agencies**: Track fleets with specific attributes (e.g., electric cars vs. diesel trucks).
* **Inventory Management**: Adaptable to other industries (e.g., retail product catalogs).
* **Educational Tool**: Demonstrates OOP principles like encapsulation and polymorphism.

**References**

1. **Oracle Java Documentation**
   * Oracle. (2023). *Interfaces*.  
     <https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>
   * Oracle. (2023). *Enum Types*.  
     <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>
2. **Input Validation & Error Handling**
   * Baeldung. (2024). *Exception Handling in Java*.  
     <https://www.baeldung.com/java-exceptions>
3. **File I/O Operations**
   * GeeksforGeeks. (2023). *File Handling in Java*.  
     <https://www.geeksforgeeks.org/file-handling-in-java/>
4. **ANSI Escape Codes**
   * Wikipedia. (2024). *ANSI Escape Codes*.  
     <https://en.wikipedia.org/wiki/ANSI_escape_codes>