OOP Project Description

Requirements:

You are to design prototype software (stand-alone application loaded on business's local computer) to manage vehicle transactions for vehicles that can be rented leased, or bought for the business. Your software will be used and called by User Interface (UI) that someone else will be designing.

Your software will allow loading and managing vehicle transactions for the business. When the system starts up, the system will load the most current transaction from the default directory or load a specific transaction based on provided transaction id.

For the prototype the transaction data will be loaded from and saved in text files on the local system (for the final product outside the scope of this project, server with database will be used). UI will not know the location nor interact directly with the data storage. Your system will manage all the data and it has to update the files as needed to support the expected functionality. The UI will need capability to download specific transaction to a local file.

The user can initiate one of three types of transactions: lease a vehicle, rent a vehicle, or buy a vehicle. A single transaction may contain multiple vehicles.

Each transaction will include a unique transaction id generated by UI (String), customer data, vehicle(s), transaction price, transaction start date (when the lease, rent, or buy will take effect), and transaction activation date (when transaction was moved from draft to active).

New transaction must have id, start date, customer data, and vehicle(s) at creation and id cannot be ever changed.

Rent transaction will also have number of days for the rental; lease transaction will also have the number of months that the lease will last, and buy transaction will also have the price of extended warranty. All of these values must be set at transaction creation. The data can only be changed when transaction is in draft mode.

<u>Note:</u> For this system design, each transaction type should be designed as a separate Java class.

When a transaction is first created, it is in draft mode and the system will set the transaction price only once the transaction is activated. Vehicle data and start date can only be changed while the transaction is in draft mode.

Vehicle data includes make, model, 17 character VIN, 4 digit year, type (0 for Sedan, 1 for Truck/Van, and 2 for SUV), and price. Vehicle cannot exist without these values and they cannot be changed. Vehicle will have capability to load from a file or by passing all the values.

Customer data will include first name, last name, mailing address, phone number, and email. All the data must have values at creation but it can be changed for draft transaction.

When user requests the transaction to be activated, the system will change the transaction state to active (as opposed to draft) and it will update the transaction activation date (that is when payment will be processed however it is out of the scope of this system) and transaction price. A transaction cannot be activated if it is past transaction start date. No additional changes can be made to an activated transaction.

To calculate transaction price:

- Base transaction price is $19.99
- Buy transaction: base price + car price + warranty + tax fee (5% of the price)
- Lease transaction: base price + 1% of vehicle price * number of months of the lease
- Rent transaction: base price + number of rental days * fee (Sedan: 39.99, Truck/Van 49.99 and SUV 42.99)

In addition, the system must support the following functionality:

- Load transaction for a specific transaction id from default directory
- Load transaction from a provided file
- Retrieve loaded transaction object
- Activate loaded transaction on a specific date
- Calculate and return transaction price
- Add vehicle to loaded transaction
- Remove vehicle from loaded transaction
- All classes except manager should have toString() which returns formatted data that are used to write to the console or a file.

Vehicle and transaction files must be human readable text files and design must follow separation of concerns principle. They should use XML or JSON format.

The system must also include the following user defined exceptions:
1. InvalidLoadException: Used for issues with transaction, such as file does not exist or there is any issue parsing it. The exception will extend RuntimeException. The exception message should indicate the filename and what the detected issue was.
2. InvalidVehicleException: Used when there are issues with file/data when adding/loading/removing vehicle. The exception will extend RuntimeException.

The exception message should indicate what operation was attempted and what/why failed. Constructors must get all the meaningful data to generate exception message.

UI is outside the scope of your design but your system needs to have all the functionality it would need to satisfy the requirements.

As you design the system, you need to make sure it is consistent and it only has the interfaces (constructors, attributes and methods) that are needed by the User Interface (UI) and to satisfy the above requirements.