An Object-Oriented Car Dealership System Part-2

Workbook 5's Workshop

Project Description

In this project, you will add Sales and Leasing to your dealership. This means you will need to add one more option to your menu - SELL/LEASE A VEHICLE.

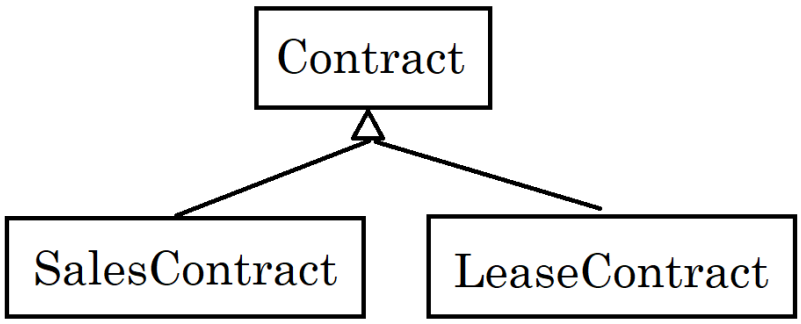
When the user records a sale or a lease, you will need to:

- collect basic sales information from the user
- add the vehicle information to the contract
- ask if it is a sale or lease (Note: you can't lease a vehicle over 3 years old)
- calculate pricing

The difference between a sale and a lease is based on how pricing is determined.

Project Details

To get this part working, you will need to add the following classes:



A `Contract` will hold information ***common to all contacts***. It should be an **abstract class** as you can't create a generic contract.

- Date (as string) of contract
- Customer name
- Customer email
- Vehicle sold
- Total price
- Monthly payment

Methods will include a constructor and getters and setters for all fields except total price and monthly payment. You should define abstract methods for `getTotalPrice()` and `getMonthlyPayment()` that will return computed values based on contract type.

A `SalesContract` will include the following additional information:

- Sales Tax Amount (5%)
- Recording Fee ($100)
- Processing fee ($295 for vehicles under $10,000 and $495 for all others
- Whether they want to finance (yes/no)
- Monthly payment (if financed) based on:
    - All loans are at 4.25% for 48 months if the price is $10,000 or more
    - Otherwise they are at 5.25% for 24 month

Methods will include a constructor and getters and setters for all fields except total price and monthly payment. You should provide overrides for getTotalPrice() and getMonthlyPayment() that will return computed values based on the rules above. It is possible that getMonthlyPayment() would return 0 if they chose the NO loan option.

A `LeaseContract` will include the following additional information:

- Expected Ending Value (50% of the original price)
- Lease Fee (7% of the original price)
- Monthly payment based on
    - All leases are financed at 4.0% for 36 months

Methods will include a constructor and getters and setters for all fields except total price and monthly payment. You should provide overrides for `getTotalPrice()` and `getMonthlyPayment()` that will return computed values based on the rules above.

Persistence Issues

When a user selects the sale or lease option, they will have to provide the VIN of the vehicle they are interested in. Once you complete the data collection for the sale or lease, you will need to APPEND it to a contracts file. If you support multiple dealerships, all contracts still go in the same file.

Example:
(Word wrap is only in this document - a sample data file is also provided for you)

```
SALE|20210928|Dana Wyatt|dana@texas.com|10112|1993|
    Ford|Explorer|SUV|Red|525123|995.00|
    49.75|100.00|295.00|1439.75|NO|0.00
LEASE|20210928|Zachary Westly|zach@texas.com|37846|2021|
    Chevrolet|Silverado|truck|Black|2750|31995.00|
    15997.50|2239.65|18337.15|541.39
```

This means you will need to create a DIFFERENT data manager class. Name it `ContractFileManager` and add a method to save the contract by APPENDING it to your contracts file.

Programming Process

We suggest you spend the first few minutes porting this project over to IntelliJ and adopting Java packages. There are many ways to go about this, but if you are unsure, we suggest:

1. Create a new GitHub repository for this workshop and clone it to your `C:\LearnToCode-Workshops` folder
2. Launch IntelliJ and create a new Java project (adv-dealership-project)
3. Create a package for the source code (com.yearup.dealership)
4. Copy your .java files from your existing car dealership project to your new `src\main\java\com\yearup\dealership` folder
5. Modify each class to have a package statement as the first line
6. Build and test your program to make sure it works
7. Commit and push your changes.

Now you are ready to begin!

Phase 1

Phase 1 should be the construction of your new data model classes: `Contract`, `SalesContract`, and `LeaseContract`.

Phase 2

Phase 2 should be the construction of your `ContractDataManager`. The `saveCotract()` method will accept a Contract parameter, but you will need to use `instanceof` to check the type of contract because the format of what you write to the file changes depending on the contract type.

Phase 3

Phase 3 involves the updating of the UserInterface. Add a new command option to gather the input from the user, then instantiate the correct type of contract and send it to ContractDataManager to be saved. Don't forget to remove the vehicle from inventory afterwards!!

At this point you can test and debug!

Bonus Ideas

If you finish early, there are two extensions you can add to this project:

1. Add an Admin command to your menu and require the user enter a password to move forward. Instantiate an `AdminUserInterface` object and call its display. Provide features to examine contracts.

For example LIST ALL CONTRACTS or LIST LAST 10 CONTRACTS.

This option will require you to create a new class to manage the user interface (not because it has to happen, but because it's a fun idea). It will also mean your `ContractDataManager` will have to be able to read in and return and `ArrayList<Contract>` objects.

2. Add the ability for a buyer to select from a set of `AddOn` objects when they *buy* a vehicle. You can hard code the `AddOn` data, but I envision it containing at least options like the following:

Nitrogen tires	Window tinting
All-season floor mats	Splash guards
Cargo tray	Wheel locks

Price them however you want. But remember, this option complicates the coding of the `ContactFileManager` -- both writing a reading. A suggestion would be to add it to the end of the contract data.

What Makes a Good Workshop Project?

- **You should:**
  - Have a clean and intuitive user interface (give the user clear instructions on each screen)
  - Implement the ability for a customer to add/remove items to a cart and also to purchase the items in the cart

- **You should adhere to best practices such as:**
  - Create a Java Project that follows the Maven folder structure
  - Create appropriate Java packages and classes
  - Class names should be meaningful and follow proper naming conventions (PascalCase)
  - Use good variable naming conventions (camelCasing, meaningful variable names)
  - Your code should be properly formatted easy to understand
  - use Java comments effectively

- **Make sure that:**
  - Your code is free of errors and that it compiles and runs

- **Build a PUBLIC GitHub Repo for your code**
  - Include a README.md file that describes your project and includes screen shots of
    * your home screen
    * your products display screen
    * one calculator page that shows erroneous inputs and an error message.
  - ALSO make sure to include one interesting piece of code and a description of WHY it is interesting.