**Proposed project title:**

Rent-a-car

**Description of project:**

I am proposing an Object-Oriented Software for vehicle rental companies. This project is essentially about the transactional data and what vehicles are available and what vehicles are checked out, however there are additional considerations such as user types for the system and what permissions they’re allowed. I envision an abstract user class with admin, owner, manager, employee, and customer extending it. Ideally, this system will handle the transactions and rental deadlines, but I will be satisfied if I can create a fully operational system in which I am able to track numerous metrics a business typically would as well as manually adjust and trigger deadlines.

**Intended User/Use Case Analysis:**

There are a couple of intended users of this software. First would be an employee at a rental business, they will access it to deal directly with customers by making a transaction, check inventory, and check rental schedules. The next user is the manager. The manager will use this to conduct the same functions as the employee, with the addition of adding new vehicles, profit tracking and employee breakdowns. The third user is the admin who will be able to add and edit users, locations, and managers. Finally, is the owner. The owner will be able to track all of this across every location.

**What Problem is the Program Trying to Solve?**

Self-explanatory on a large scale. On a small scale, the problems needing to be solved involve array persistence and manipulation when numerous differing objects are involved. Ideally, as this is just a project, the scope doesn’t grow to be unmanageable as to make a ‘business-ready’ application is certainly beyond the scope. I’d say overall the problem needing solved is barebones business operations for a vehicle rental business, with the ability to add functioning expansion locations on, without sacrificing functionality.

**Technologies:**

Files mostly, if maintaining serialization, still undecided as to the overall inner-workings and whether or not a database would be a better approach.

**Concepts:**

**Memory Allocation:** Instances of employees will be allocated on the heap as an arraylist, as will the vehicle inventory and any other instances as needed.

**OOP:** Making in java, everything will be objects

**Abstraction:** Usage of abstract classes vehicle class, user class, transaction class

**Encapsulation:** All member variables managed/navigated by getters&setters

**Inheritance:** Subclasses inheritmethods and variables from parent class

**Polymorphism:** transaction class is able to take multiple parameters and adjust functionality accordingly

**Persistence:** Object serialization for data persistence.

**Data Structures:**

ArrayList: Employee objects and vehicle objects

**Milestones:**

.5 All classes built

1. Basic transaction and file IO inventory tracking through serialization

2. Add/edit employees with admin

3. Differentiation between admin features and manager features

3. multi location adaptability (users log into their location’s system only)

3.5 Unless user is owner or admin

3.5.5 Then owner or admin chooses location

Admin assigns users and managers to location

4. Owner access to list employees/profit/etc by location or overall list

------------------------------------------------------------------------------------------------------------------------------------------

**Algorithm:**

Classes:

User

* Customer
* Employee
  + Manager
  + Owner
  + Admin

Vehicle

* Subclasses(?)

Transaction(?)

Company

* Placeholder location

Date

Menu

---------------------------------------------------------------------------------------------------------------------

**User:**

Getters and setters and constructor for name and type variables

**Employee (extends User)**

**checkInventory()/checkLoans():**

Goals: To iterate through the arrays and print what is in them

Input: N/A

Output: Whatever is in the arrays printed to console

Steps:

1. For each object in array, print year, make, and model.
2. For each on loan, print above, but also customer info and rent/return dates

**Admin (Extends Owner)**

Adds and deletes for various users and locations.

Just constructor call and add to respective array list

**Company**

**listLocations()**

Goal: Return a list of locations where the business has offices and return various data about each location

Input: N/A

Output: Console list of information

Steps:

For each location in arraylist, return name and address

**addLocation():** adds a location to array list

**login()**

Goal: To provide a login menu which prompts for credentials and passes them to verification function

Input: Prompt for input

Output: Input to verification function

Steps:

1. Ask for username and password
2. Call verification function with input variables as parameters

**verifyCredentials(username, password)**

Goal: To verify the credentials input by a user

Input: The parameters input and password

Output: A Boolean T or F

Steps:

1. If the input credentials belong to admin, bring up admin menu
2. Else look through the employee array and compare credentials
3. If found, return true, else return false

**Location:**

Standard getters and setters for all variables, nothing new with the array list operations.

**addTransactionRecord()**

Goal: To output a transaction to an external file

Input: Still figuring this part out

Output: Information for the transaction (customer info, vehicle info, cost, duration, location, etc.) to an external file (in append mode)

Steps: Depends on the input. Still working on this

**loanVehicle(Vehicle, Customer):**

Goal: To serve as a method of transaction to assign a vehicle to a user and remove it from the available vehicles and place it in loaned vehicles

Input: A customer object and a vehicle object

Output: Nothing really output, but data is manipulated

Steps: Still working on specifics

**reclaimVehicle(Vehicle, Customer):**

Opposite of loanVehicle above, how this works exactly depends on how the above works.

Diagram

Description automatically generated