**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, and create and edit customers. The next user is the admin who will be able to add and edit users, and locations.

**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 user class to create various user types

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

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

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

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

**Data Structures:**

ArrayList: Employee objects and vehicle objects

**File IO:**

Transaction log is saved and all transactions are written to an external log document

**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(Abstract):**

Getters and setters and constructor for name and type variables

**Employee (extends User)**

**Admin (Extends User)**

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

**loginVerify(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

**menuEvaluator()**

checks isUserAdmin and if true opens adminMenu, if user isn’t admin opens employee menu

**Menus:**

mainMenu(): provides option to login or exit program

* logIn()

adminMenu(): gives access to admin functions to manage employees, inventory, and locations.

* manageEmployeesMenu()
* manageInventoryMenu()
* manageLocationMenu()

manageEmployeesMenu(): add remove or list employees

* addEmployee()
* deleteEmployee()
* list employees (not a function, it’s a for-each loop)

manageLocationMenu(): add, delete, or list locations

* addLocation()
* deleteLocation()
* lists locations (not a function, a for each loop)

manageInventoryMenu(): add, delete, or list vehicles

* addVehicle()
* deleteVehicle()
* lists vehicles at each location, by location (for each loop)

customerManagementMenu(): add, delete, or list customers

* addCustomer()
* deleteCustomer()
* listCustomers()

transactionMenu(): takes a parameter from transactionSubMenu() in the form of a String specifying if transaction is incoming or outgoing. Moves vehicle to a customer and to the rented array, or takes back vehicle from customer and moves vehicle to ‘available’ array

transactionSubMenu(): evaluates whether a customer is renting or returning a vehicle and passes the typeTransaction variable as a parameter to the transactionMenu() function

employeeMenu(): manage customers, inventory, or make a transaction

* customerManagementMenu()
* manageInventoryMenu()
* transactionSubMenu()

**logTransaction():**

transactions are recorded to a text document to log all transactions. Employee conducting the transaction, the Vehicle involved, and the Customer is recorded.

**addEmployee():**

User fills in the requested parameters and a new Employee is created and added to the employee array

**deleteEmployee():**

user selects an employee by number from a listing of employees. The number is passed to the remove method of arrayList and the employee is removed from the array

**addVehicle():**

Asks user what location the vehicle will go to andcalls createVehicle functions and adds the created vehicle object to the specified location’s vehicle available arraylist

**deleteVehicle():**

user selects the location and then the vehicle from prompts. The vehicle is deleted by passing the user input number to the arrayList remove() method

**add/deleteCustomer():**

same as the above. User is walked through prompts to create the properties of Customer and then customer is added to arrayList for customers

**Create Functions:**

Address, Date, Vehicle, Payment objects are created using prompts from the system to fill in needed parameters. Used to save code space for repeated creation functions

**Location:**

Standard getters and setters for all variables, nothing new with the array list operations. Also functions to iterate through arraylists and print what is in them. Add and Remove functions are also included for array list operations.

