Robert Houst 04/04/2024 CS3331- Advanced Object-Oriented Programming-Term Spring 2024 Bhanukiran Gurijala Project Part 1

### 1. Program Explanation:

The task is to develop a management system for auto dealerships. Users of the system can purchase vehicles, view tickets, filter cars, view cars, log in, and sign out. It also has administrative features for logging actions and updating vehicle information. The user and vehicle data kept in CSV files must be read and updated by the system. I first determined the primary functionalities needed by analyzing the requirements in order to approach this problem. To see how the actors—users and administrators—interacted with the system, I next constructed a UML Use Case Diagram (Level I).

I Ire able to comprehend the system's general flow thanks to this.

To organize the code, I created a UML Class Diagram next. I made classes using the main method for users, admins, automobiles (including special kinds like sedans and SUVs), logs, and cars. Every class included fields and methods that Ire suitable for handling the necessary capabilities.

I made use of the CSV parsing libraries that come with Java to handle reading and storing data from CSV files. In order to read and update the CSV files in response to the user's activities (such as buying a car or revising their budget), I built methods in the User and Car classes.

I divided the issue into more manageable subissues, like these:

- putting in place login and user authentication.
- putting in place the browsing, filtering, and buying capabilities for cars.
- putting in place user activity logging.
- updating data in CSV files about the user and the vehicle.

### 2. What did I learn?

I gained knowledge on how to use Java programming and UML diagrams to build and implement an automobile dealership management system. Additionally, I gained knowledge of setting up user identification, managing user interactions via a menu-driven interface, and reading and updating data from CSV files.

Several things could be done to make your solution better:

- Handling errors: To address situations like incorrect user input, file not found issues, etc., include more reliable error handling.
- Code modularity: To make your code easier to read and maintain, divide it up into smaller, more manageable functions and classes.
- Improve the user interface to make it more intuitive and user-friendly.
- Use data validation to make sure that only legitimate inputs are accepted.

Taking a lb-based approach to the problem is an additional solution. Using HTML, CSS, JavaScript, and a backend framework like Spring Boot or Node.js, you may develop a lb application. This method would make the system more scalable and accessible by enabling users to access it using a lb browser.

### 3. Solution Design

I created and put into use a Java auto dealership management system for this program. Users of the system can purchase automobiles, explore and filter cars, view tickets, and sign out. It also has administrative features for logging actions and updating vehicle information. The software reads and updates data from CSV files that contain user and vehicle information.

I started by analyzing the requirements and determining the primary functionalities required in order to solve this problem. I then created Use Case and Class diagrams using UML to show the organization and dynamics of the system. I used Java to develop the system, concentrating on writing modular and reusable code, based on the diagrams.

I used CSV files to store user and vehicle data for data storage. I read and updated these files using the CSV parsing libraries that come with Java. I also managed and stored items in memory, like user and vehicle instances, using collections like ArrayLists. These data structures Ire selected due to their ease of use and effectiveness in handling the necessary data.

Our presumptions consist of:

- The user and vehicle data are contained in III-formatted, error-free CSV files.
- Every person has a distinct username.
- A single instance of the program can only have one user logged in at once by the system.
- The user CSV file stores the budget for buying an automobile as a double value.

## 4. Testing

I combined white-box and black-box testing methods to test the program. Testing a program's functionality without being aware of its internal workings is known as "black-box" testing. This involved testing the login, car-browsing, car-filtering, car-purchasing, ticket-viewing, and sign-out features.

White-box testing entailed analyzing the logic and internal workings of the program to make sure it operated as intended. This involved testing the logging and reading of CSV files in addition to their updating.

Although I think I tested the solution thoroughly, testing procedures can always be improved. In order to enhance testing, I may have:

- To cover a wider range of scenarios, including edge cases and invalid inputs, the number of test cases was increased.
- To guarantee consistency and effectiveness in testing, automated testing was conducted utilizing tools such as JUnit.

- conducted stress testing to evaluate the program's performance with big datasets or under high load.
- strengthened error handling and validation procedures to detect and manage unforeseen inputs or circumstances.

## Among the test cases I employed are:

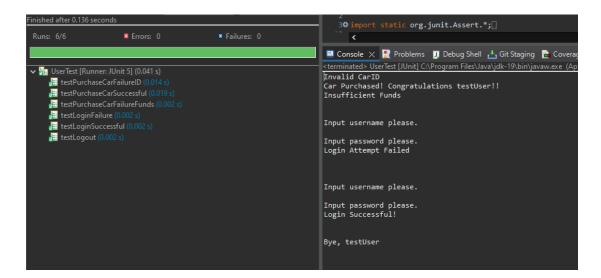
- entering a working username and password to log in.
- using an incorrect password or username to log in.
- looking through every car and making sure the right information is shown.
- separating cars into different categories (used/new) and making sure the right vehicles are shown.
- buying a car and making sure the amount of cars is reduced and the user's budget is updated.
- trying to buy an automobile on a shoestring.
- checking sure the right ticket information is displayed by looking at the tickets.
- logging off and making sure the person has left their account.

While I didn't intentionally break the software to make it better, I did take into account edge cases and invalid inputs during testing to make sure the program handled them III.

### 5. Test Results

# User verification (sign in)

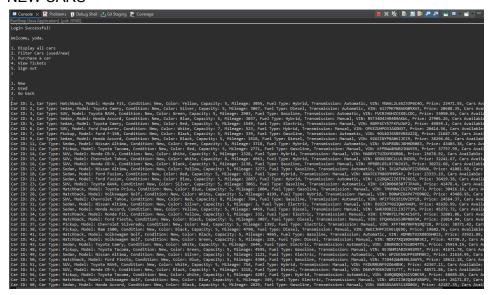
- Test: To confirm login functionality, enter both valid and incorrect credentials.
- Anticipated outcome: If the credentials are legitimate, the login will be successful; if not, it will fail.
- Console Output: Show a message indicating if the login attempt succeeded or failed.



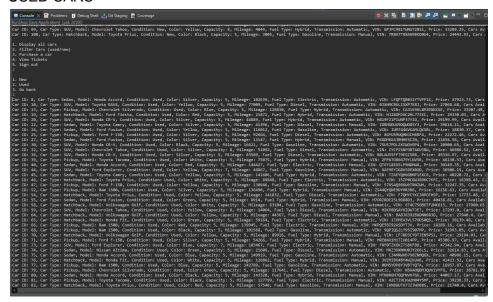
# **Driving and Reviewing**

- Test: Show every car and sort by new or used.
- Anticipated Outcome: A list of vehicles sorted by filter displayed.
- Console Output: Show a list of vehicles together with their attributes.

# **NEW CARS**



# **USED CARS**



### **ALL CARS**

```
**Control Mark Robbons*** Discharge **Coverage **Covera
```

# **Buying a Car**

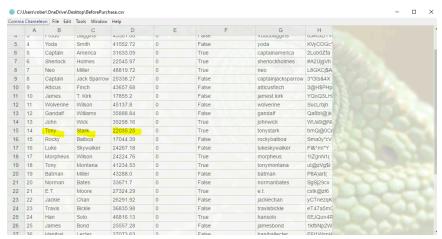
- Test: Try buying a car both with and without enough money.
- Anticipated outcome: If funds are available, the transaction will be successful; if not, it will fail.
- Console Output: An error message indicating insufficient budget and a
  confirmation message upon successful purchase. (You might want to add
  somewhere that the test for purchasing is only successful if the users
  moneyAvailable is adjusted to an appropriate predetermined value, the size of
  the users tickets array list is increased by 1 and the users carsPurchased
  variable is increased by 1.)

```
Car ID: 96, Car Type: Pickup, Model: Toyota Tacoma, Condition: Used, Color: Red, Capacity: 5, Mileage: 36178, Fuel Type: Diesel, Transmission: Automatic, VIN: MOIZIBMFCMRBTADSG, Pr
1. Display all cars (used/new)
3. Purchased a car
4. View Tickets
5. Sign out
5
Car Purchased: Congratulations tonystark!!
1. Display all cars
2. Filter Cars (used/new)
4. View Ticket ard
4. View Ticket ard
5. Sign out
5. Purchased: Congratulations tonystark
1. Display all cars
2. Filter Cars (used/new)
3. Purchased: Congratulations tonystark
1. Displaying all user tickets for: tonystark
1. Displaying all user tickets for: tonystark
1. Display all cars
2. Filter Cars (used/new)
3. Purchased: (used/new)
5. Sign out
5. Sign out
5. Sign out
5. Sign out
6. Sign out
7. Purchased: Sign out
8. Sign out
9. Sig
```

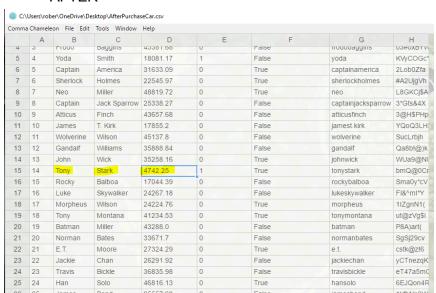
## Modify the user's CSV

- Test: Add the updated budget to the user's CSV file.
- Anticipated Outcome: A CSV file with the updated budget should be the expected outcome.
- Text Document Result: The modified user's CSV file contents.

### **BEFORE**



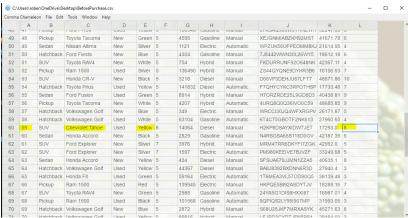
### **AFTER**



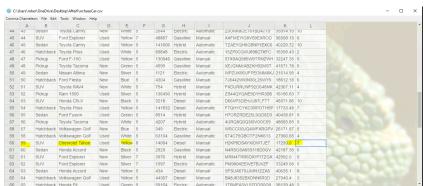
## Update the CSV for the car.

- Test: Add the updated number of automobiles available to the car's CSV file.
- Anticipated Outcome: The new quantity ought to be reflected in the CSV file, as anticipated.
- Text Document Result: Contents of the CSV file containing the modified automobile.

# BEFORE



### **AFTER**



### **Recording Activities**

- Test: Run through a series of actions and see whether any logs are generated.
- Anticipated Outcome: Records have to include details regarding the actions executed.
- Text Document Outcome: The log file's contents.

```
File Edit View

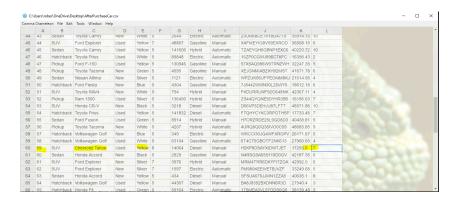
Property Session Logged
Thu Apr 04 20:57:22 MDT 2024 - Car data read by system
Thu Apr 04 20:57:22 MDT 2024 - System generated Car Sheet successfully File Name: car_data_snapshots_20-57-22_04_04_24.csv
Thu Apr 04 20:57:22 MDT 2024 - System generated User Sheet successfully. File Name: user_data_snapshot_20-57-22_04_04_24.csv
Thu Apr 04 20:57:22 MDT 2024 - System generated User Sheet successfully. File Name: user_data_snapshot_20-57-22_04_04_24.csv
Thu Apr 04 20:57:22 MDT 2024 - System generated User Sheet successfully
Thu Apr 04 20:57:23 MDT 2024 - System loaded user tickets successfully
Thu Apr 04 20:57:31 MDT 2024 - System loaded user tickets successfully
Thu Apr 04 20:57:31 MDT 2024 - tonystark Attempted to purchase car with ID: 9 - INSUFFICIENT FUNDS
Thu Apr 04 20:57:34 MDT 2024 - tonystark Attempted to purchase car with ID: 72 - INSUFFICIENT FUNDS
Thu Apr 04 20:57:44 MDT 2024 - tonystark Attempted to purchase car with ID: 25 - INSUFFICIENT FUNDS
Thu Apr 04 20:57:45 MDT 2024 - tonystark Attempted to purchase car with ID: 55 - INSUFFICIENT FUNDS
Thu Apr 04 20:57:45 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase car with ID: 65 - INSUFFICIENT FUNDS
Thu Apr 04 20:58:18 MDT 2024 - tonystark Attempted to purchase ca
```

## **Exit Program**

- Test: Close the application and check the modified CSV files.
- Anticipated Outcome: Any modifications performed during the session expected to be reflected in the CSV files.
- Text Document Result: After the software is closed, the data of the updated CSV files for the user and the car.

### **USER**

Commi	Chamele	on File Edit	Tools Window	Help				
	А	В	С	D	E	F	G	Н
4	0	FTOUO	Dayyırıs	40301.00	U	Fdise	поцорадушь	U3#UXDTV
5	4	Yoda	Smith	18081.17	1	False	yoda	KVyCOGc
6	5	Captain	America	31633.09	0	True	captainamerica	2Lob0Zfa
7	6	Sherlock	Holmes	22545.97	0	True	sherlockholmes	#A2UjgVh
8	7	Neo	Miller	48819.72	0	True	neo	L8GKCj\$A
9	8	Captain	Jack Sparrow	25338.27	0	False	captainjacksparrow	3*Gts&4X
10	9	Atticus	Finch	43657.68	0	False	atticusfinch	3@H\$PHp
11	10	James	T. Kirk	17855.2	0	False	jamest.kirk	YQ0Q3LH
12	11	Wolverine	Wilson	45137.8	0	False	wolverine	SucLrbjh
13	12	Gandalf	Williams	35888.84	0	False	gandalf	Qa8bl@)k
14	13	John	Wick	35258.16	0	True	johnwick	WUa9@N
15	14	Tony	Stark	4742.25	1	True	tonystark	bmQ@0C
16	15	Rocky	Balboa	17044.39	0	False	rockybalboa	Sma0y*cV
17	16	Luke	Skywalker	24267.18	0	False	lukeskywalker	FI&^mI*Y
18	17	Morpheus	Wilson	24224.76	0	True	morpheus	1IZgnN1(
19	18	Tony	Montana	41234.53	0	True	tonymontana	ut@zVg\$i
20	19	Batman	Miller	43288.0	0	False	batman	P8A)art(
21	20	Norman	Bates	33671.7	0	False	normanbates	SgSj29cv
22	21	E.T.	Moore	27324.29	0	True	e.t.	cstk@zl6
23	22	Jackie	Chan	26291.92	0	False	jackiechan	yCTnezqk
24	23	Travis	Bickle	36835.98	0	False	travisbickle	eT47a5m
25	24	Han	Solo	46816.13	0	True	hansolo	6EJQon4F
200	25	lamas	Dond	05557.00	0	Foloo	iomonhond	4 lefts N to ON



### 6. Code Review

## **Quality of Code**

- *Verify Consistency*: Make sure the codebase's names, layout, and programming style are all the same.
- Verify the readability: the code by making sure it is clear and, if needed, Ill-commented.

## **Operational**

- Examine Requirements: Verify that the code satisfies the functional specifications and requirements as outlined in the use case and class diagrams.
- Test Cases: Ensure that the test cases pass and cover all areas of the code.

### **Achievement**

- Optimization: Look for any places where the code could be made more efficient or for any performance bottlenecks.
- Resource Usage: Verify that resources like CPU and memory are being used effectively.

# Safety

- Verify: that every user input is validated to stop security flaws
- Authentication and Authorization: Check to make sure the procedures for authorization and authentication are set up properly.

## **Error Reduction**

- Verify Exception Handling: Make that exceptions are managed properly and politely.
- Logging: Ensure that errors and important events are recorded in logs.

# Record-keeping

• Look for Comments: Make sure the code has clear documentation that explains any complicated logic or features.

# **Testing**

- *Test Coverage:* Verify that there is enough test coverage to cover every important section of the code.
- Testing for Regression: Ensure that modifications don't cause new issues or errors.

# Maintainability and Flexibility

- *Verify*: that the code is simple and follows accepted practices for maintainability and durability.
- Code Duplication: Look for instances of duplicate code and, if needed, rework it.