Nara Macias, Sofia Martinez, Jenna Murphree
April 20, 2024
CS 3331 – Advanced Object-Oriented Programming – Spring 2024
Dr. Gurijala
Project Part II Report

This work was done individually/as a team and completely on my/our own. I/we did not share, reproduce, or alter any part of this assignment for any purpose. I/we did not share code, upload this assignment online in any form, or view/received/modified code written from anyone else. All deliverables were produced entirely on my/our own. This assignment is part of an academic course at The University of Texas at El Paso and a grade will be assigned for the work I/we produced.

1. Program Explanation

<u>In this section, explain the overview of the assignment.</u>

We expanded "Mine Cars," our car dealership system, to accommodate new requirements aimed at enhancing functionality and user experience. This involved implementing features such as handling variability in input file formats, introducing interfaces and design patterns, empowering administrators with additional functionalities, refining the purchase process, enabling new user capabilities, and ensuring correct data handling.

What did you do?

- We enhanced "Mine Cars" to handle input files with non-standard column orders, ensuring seamless data ingestion.
- We implemented the Factory Method pattern through Factory<T> interfaces, enabling dynamic creation of Car, User, and Ticket objects, which streamlined our code's modularity and facilitated future enhancements.
- Leveraging the Factory Method design pattern, we refactored our code to dynamically create objects, enhancing flexibility.
- New admin functionalities were added, including inventory management, revenue tracking, and user management.
- The purchase process was updated to incorporate membership discounts and state tax calculations, enhancing user incentives and regulatory compliance.
- Users were empowered with the ability to return purchased cars, improving transactional flexibility and customer satisfaction.
- Data handling mechanisms were implemented to gracefully manage empty cells in CSV files, ensuring accurate processing of vehicle attributes.

How did you tackle the problem?

- We approached each requirement systematically, breaking down complex tasks into manageable components.

- Utilizing our understanding of software design principles and design patterns, we devised elegant solutions to address diverse challenges. For instance, we scrutinized our approach to creating cars within the system, ensuring we avoided hardcoding and instead focused on a dynamic, reusable methodology.
- We made sure each class handled distinct aspects of the dealership system, such as user interactions, car data management, or transaction processing.

What techniques did you use to solve the problem?

- We employed structured planning, in-depth analysis, and team collaboration to tackle each requirement effectively.
- Techniques included dynamic object creation using the Factory Method pattern, interface implementation for modularity, and resilient data handling strategies.
- Refactoring existing codebase to accommodate new features and design patterns was a key technique employed to ensure scalability and maintainability.
- Testing methodologies such as unit testing and integration testing were utilized to verify the correctness of implemented solutions.
- Dynamic Object Creation - Used the Factory Method pattern to dynamically create objects, facilitating flexibility in object instantiation.
- UML and Class Diagrams - Updated UML and class diagrams to reflect the new requirements and design changes, aiding in visual understanding and documentation of the system structure.
- Interface Implementation for Modularity - Implemented interfaces to ensure modularity, allowing components to be interchanged and tested more easily.

Did you break the problem into smaller problems? Explain.

- Yes, we decomposed the overarching requirements into smaller, manageable tasks, focusing on individual components of the system. This involved splitting the tasks based on their functionalities and priorities to ensure efficient utilization of resources.
- Each requirement was analyzed independently, and a step-by-step approach was adopted to address them systematically. By dividing the work into smaller, more manageable tasks, we were able to allocate resources effectively and mitigate potential bottlenecks.
- Collaborative planning and brainstorming sessions were conducted to ensure alignment with client expectations and project objectives. During these sessions, tasks were distributed among team members based on individual expertise and availability, optimizing productivity and time effectiveness.
- By breaking down the problem into smaller sub-problems, we ensured a focused and efficient development process, leading to a comprehensive and cohesive solution. This approach enabled us to maintain momentum and deliver incremental improvements while maintaining overall project coherence.

2. What did I learn?

What did you learn as a result of this assignment?

- As a result of this assignment, we gained insights into the significance of dynamic code handling for improved maintainability. We realized the importance of incorporating robust comments and documentation, facilitating easier future modifications. Additionally, we recognized the need to validate admin inputs to ensure accurate data entry, enhancing system reliability and user experience.

How can my solution be improved?

- Improving our code by implementing validation checks on admin inputs would bolster the reliability and robustness of our system. Currently, we assume that admins will correctly enter all required information for creating new cars and users. However, introducing validation mechanisms to ensure that all necessary data fields are provided before proceeding with car or user creation would minimize the risk of incomplete or erroneous entries. By enforcing data validation at the input stage, we can preemptively address potential issues, maintain data integrity, and enhance the overall quality of our system.

What ideas do you have about another way to solve the problem?

- Database Upgrade: Transition from CSV storage to a robust database system to improve data management and scalability.
- API Development: Implement APIs to enhance system modularity and facilitate integration with other services.
- CI/CD Implementation: Introduce Continuous Integration/Continuous Deployment pipelines to streamline development processes and ensure consistent, high-quality updates.

How long did it take me to complete this lab assignment?

- The completion of this lab assignment took us approximately 30 hours of collaborative work spread over two weeks. The time invested varied among team members, reflecting our collective effort to develop and refine the "Mine Cars" system.

3. Solution Design

What did you do in this program?

- The system initialization process involves reading data from CSV files to populate the car inventory and user base. It establishes car and user objects to facilitate dealership operations efficiently. The system offers a menu-driven interface, enabling users to browse cars, make purchases, and perform administrative tasks.
- The project is well-organized into specific folders, each tailored to different functionalities:
- Models/: Houses classes like User, Ticket, Car, etc., essential for representing system entities.

- Factory/: Contains factories for creating Car, User, and Ticket objects, centralizing creation logic.
- Handlers/: Includes FileHandler for managing CSV operations and ShopManager for business logic.
- UI/: Hosts user interface logic and serves as the entry point for user interactions.
- Utils/: Contains utilities such as logging tools for system monitoring and debugging.

What was your approach to solving this problem?

- We began with Use Case Scenarios, identifying essential user interactions like login and browsing. This provided insight into user needs and system requirements. Translating these scenarios into a structured Class Diagram bridged the gap between conceptual planning and code development, ensuring a user-centric design and organized coding approach.

What data structures did you use? Why?

- Initially, we used ArrayList for its dynamic array capabilities, enabling flexible handling of CSV data and the dealership's inventory. However, we transitioned to HashMaps for their flexibility and efficient key-value storage, facilitating quick access to objects by key. This change enhanced system dynamics and maintenance ease.
- We incorporated LinkedHashMaps to preserve the insertion order of entries, which is crucial for maintaining the same order of attributes as they appear in the CSV files. This feature aids in consistent data handling and ease of system debugging and reporting.

What assumptions, if any, did you make?

- Upon further review and iteration, we updated use case scenarios to incorporate additional functionalities and user interactions.
- We maintained the assumption that admin inputs for creating new cars and users are correct, simplifying validation requirements for initial implementation.
- coIn cases where the car data files contain "null" inputs, we assumed these should be interpreted as 0, false, or the equivalent default state, depending on the field. This approach helps in handling missing data efficiently without interrupting system operations.

4. Testing

How did you test the program?

Black-box Testing:

Focus: Tested system functionality from an end-user's perspective without considering the internal workings.

Actions Tested: Included user actions like logging in, browsing cars, and processing transactions, to ensure the system responded correctly.
White-box Testing:
Focus: Examined the internal structure and logic of the code.
Specifics Tested: Covered critical functions like file handling and data processing, ensuring all logic paths and interactions between modules were correct.

Did you use black-box, white-box testing, or both? Why?
Both, This combination ensured the system was externally functional and internally robust.

Did you test the solution enough? How can the testing practices be improved? What are the test cases I used?

- System Navigation: Verified navigation flows through different system menus without unexpected exits or errors.
- User Input Validity: Tested how the system handled valid and invalid inputs for user registration and car transactions.
- Data Handling: Checked the system's ability to correctly parse and store data from CSV files and handle null or incorrect values effectively.

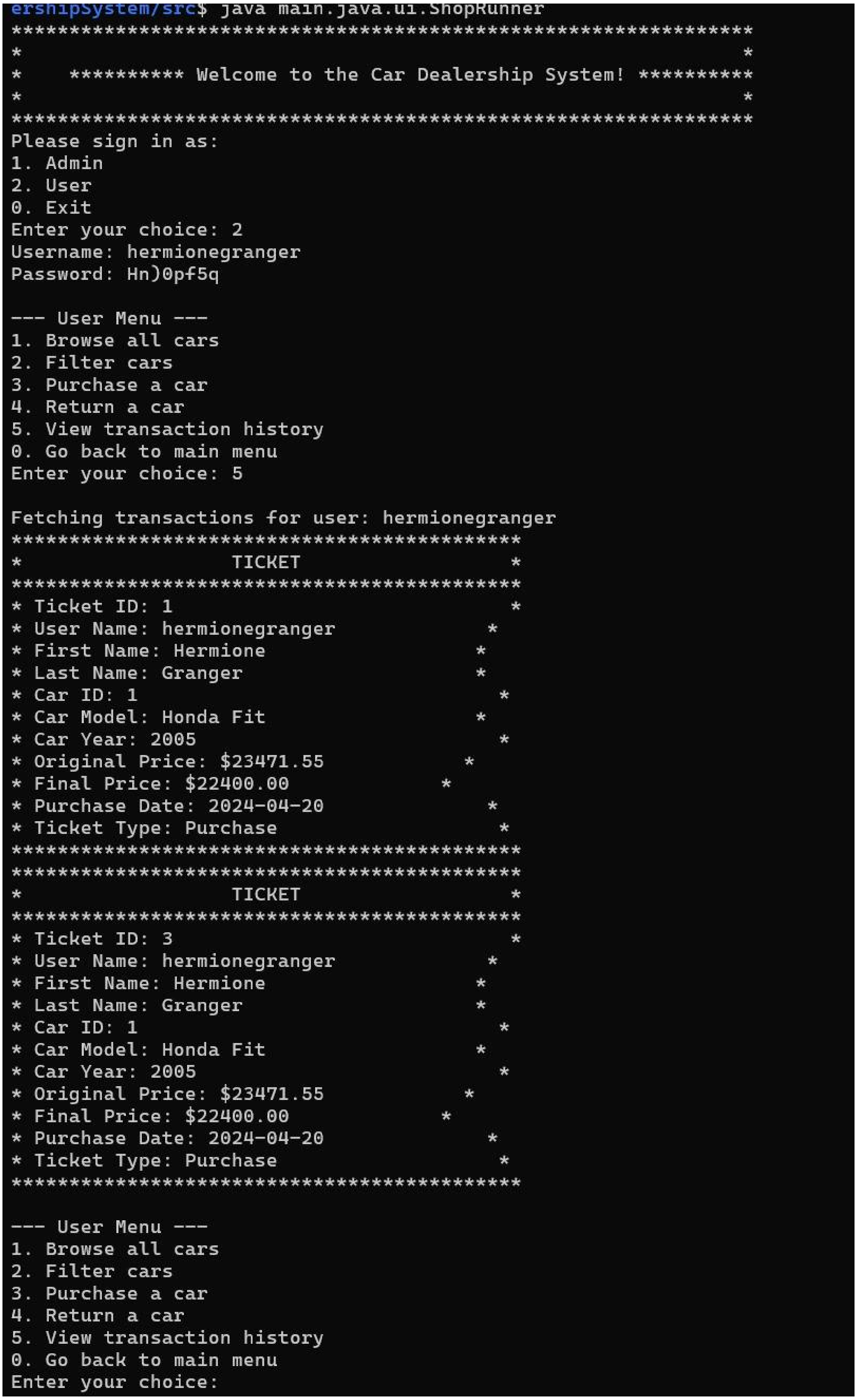
Did you break the program and use that to improve it?

- Yes, we intentionally introduced errors and abnormal inputs to break the program and test its resilience. By observing how the system handled these disruptions, we identified and rectified weaknesses, significantly enhancing its error handling capabilities and overall stability.

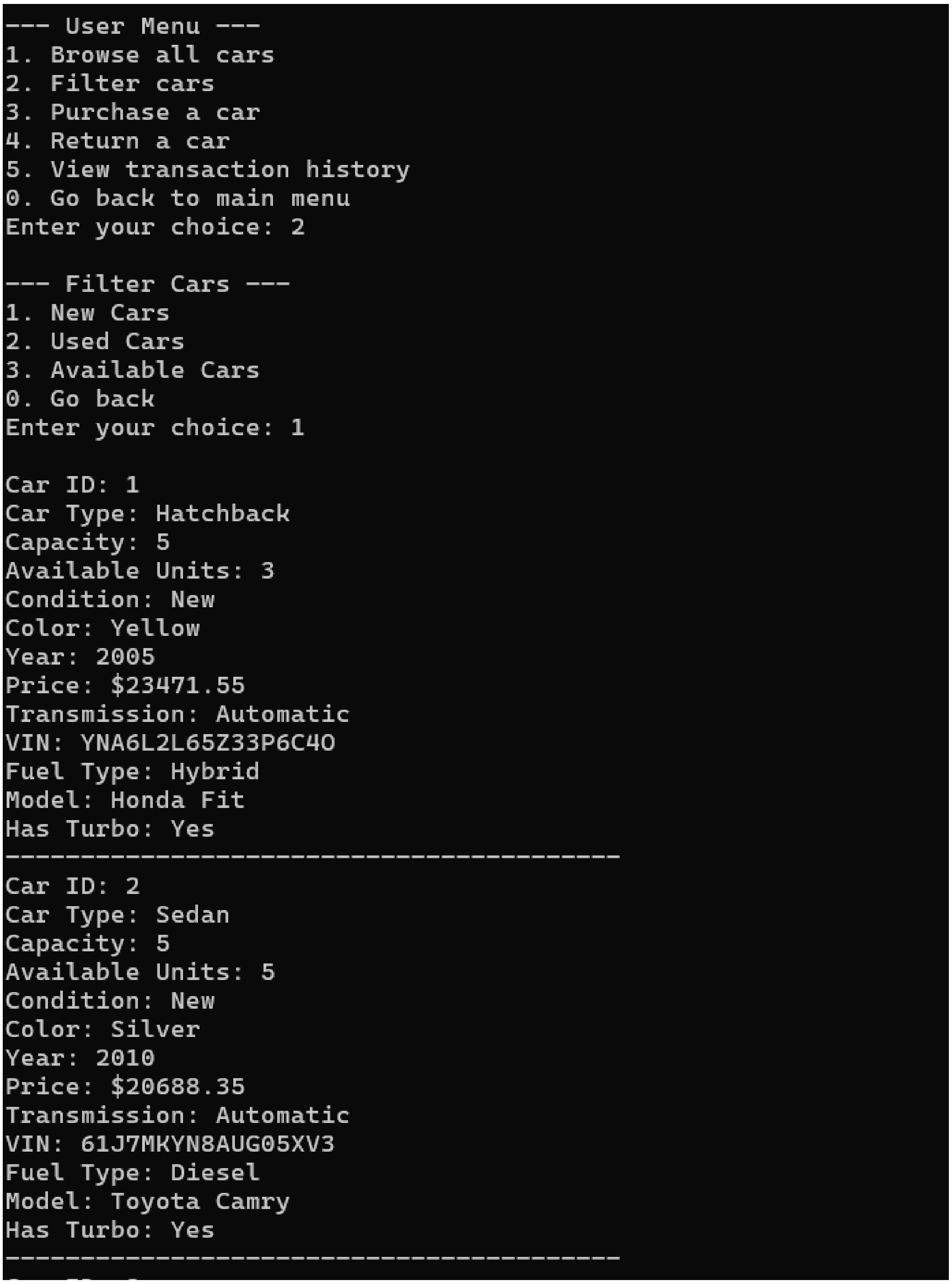
5. Test results

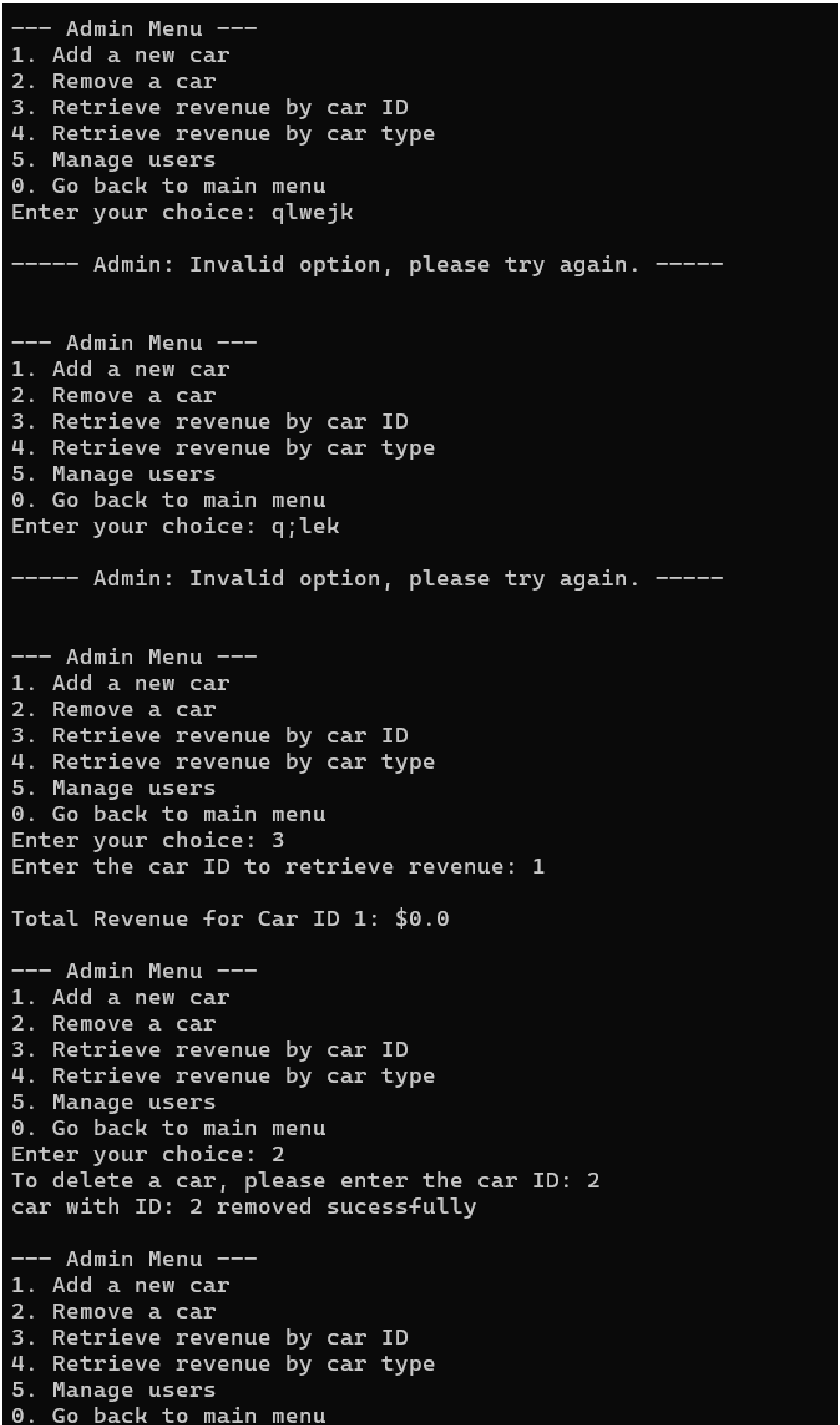
Describe the results of your tests.
We successfully tested all functions of the system, including buying, returning, and filtering operations, either through console logging or direct implementation, and the test results confirmed that each function performed as expected.

Include any console outputs showing your results.



```
ershipSystem/src$ java main.java.ui.ShopRunner
****************************************************************
*                                                              *
*    ********** Welcome to the Car Dealership System! **********
*                                                              *
****************************************************************
Please sign in as:
1. Admin
2. User
0. Exit
Enter your choice: 2
Username: hermionegranger
Password: Hn)0pf5q

--- User Menu ---
1. Browse all cars
2. Filter cars
3. Purchase a car
4. Return a car
5. View transaction history
0. Go back to main menu
Enter your choice: 5

Fetching transactions for user: hermionegranger
********************************************
*                  TICKET                  *
********************************************
* Ticket ID: 1                             *
* User Name: hermionegranger             *
* First Name: Hermione                  *
* Last Name: Granger                    *
* Car ID: 1                               *
* Car Model: Honda Fit                  *
* Car Year: 2005                          *
* Original Price: $23471.55            *
* Final Price: $22400.00             *
* Purchase Date: 2024-04-20              *
* Ticket Type: Purchase                   *
********************************************
********************************************
*                  TICKET                  *
********************************************
* Ticket ID: 3                             *
* User Name: hermionegranger             *
* First Name: Hermione                  *
* Last Name: Granger                    *
* Car ID: 1                               *
* Car Model: Honda Fit                  *
* Car Year: 2005                          *
* Original Price: $23471.55            *
* Final Price: $22400.00             *
* Purchase Date: 2024-04-20              *
* Ticket Type: Purchase                   *
********************************************

--- User Menu ---
1. Browse all cars
2. Filter cars
3. Purchase a car
4. Return a car
5. View transaction history
0. Go back to main menu
Enter your choice:
```


```
--- User Menu ---
1. Browse all cars
2. Filter cars
3. Purchase a car
4. Return a car
5. View transaction history
0. Go back to main menu
Enter your choice: 2

--- Filter Cars ---
1. New Cars
2. Used Cars
3. Available Cars
0. Go back
Enter your choice: 1

Car ID: 1
Car Type: Hatchback
Capacity: 5
Available Units: 3
Condition: New
Color: Yellow
Year: 2005
Price: $23471.55
Transmission: Automatic
VIN: YNA6L2L65Z33P6C4O
Fuel Type: Hybrid
Model: Honda Fit
Has Turbo: Yes
-----------------------------------------
Car ID: 2
Car Type: Sedan
Capacity: 5
Available Units: 5
Condition: New
Color: Silver
Year: 2010
Price: $20688.35
Transmission: Automatic
VIN: 61J7MKYN8AUG05XV3
Fuel Type: Diesel
Model: Toyota Camry
Has Turbo: Yes
-----------------------------------------
```



```
--- Admin Menu ---
1. Add a new car
2. Remove a car
3. Retrieve revenue by car ID
4. Retrieve revenue by car type
5. Manage users
0. Go back to main menu
Enter your choice: qlwejk

----- Admin: Invalid option, please try again. -----


--- Admin Menu ---
1. Add a new car
2. Remove a car
3. Retrieve revenue by car ID
4. Retrieve revenue by car type
5. Manage users
0. Go back to main menu
Enter your choice: q;lek

----- Admin: Invalid option, please try again. -----


--- Admin Menu ---
1. Add a new car
2. Remove a car
3. Retrieve revenue by car ID
4. Retrieve revenue by car type
5. Manage users
0. Go back to main menu
Enter your choice: 3
Enter the car ID to retrieve revenue: 1

Total Revenue for Car ID 1: $0.0

--- Admin Menu ---
1. Add a new car
2. Remove a car
3. Retrieve revenue by car ID
4. Retrieve revenue by car type
5. Manage users
0. Go back to main menu
Enter your choice: 2
To delete a car, please enter the car ID: 2
car with ID: 2 removed sucessfully

--- Admin Menu ---
1. Add a new car
2. Remove a car
3. Retrieve revenue by car ID
4. Retrieve revenue by car type
5. Manage users
0. Go back to main menu
```

Include any text document results of your tests.

6. **Code Review**
Explain how you conducted a review of your code. Describe how you checked each part of the code review checklist.

Implementation
Does this code do what it is supposed to?
Yes, the code manages data from CSV files, handles user and admin interactions, and maintains a dynamic system based on user inputs.

Can it be simplified?
Some parts can be simplified by abstracting repetitive tasks into methods or using more advanced collection handling.
Is the code dynamic or hardcoded?
The code is mostly dynamic but includes hardcoded paths and authentication keys that could be externalized.

Is the code maintainable?
- Yes, the modular design and separation into different packages enhance maintainability, though improvements in documentation and config management could further help.

Logic
Cases where code does not behave as expected/intended?
Error handling could be inconsistent, particularly if unexpected file formats or network issues occur.

Test cases where it may fail?
It may fail in scenarios of incorrect file formats, unexpected user inputs that aren't sanitized, or when external resources are unavailable.

Readability/Style
Easy to read/understand?
Generally, yes. The use of comments and a structured layout helps in understanding the flow of the application.

What parts can be modified or adjusted?
The handling of CSV operations and authentication could be abstracted into more generic methods to reduce redundancy.

Is the structure appropriate?

The structure is appropriate with clear module separation; however, more distinct separation between data handling and business logic could be beneficial.

Does it follow the appropriate language style?
Yes, it adheres to Java standards in naming and structuring but could improve in areas like exception handling and logging.

Is the code well documented?
There's decent documentation, but critical methods and complex logic sections could benefit from more detailed comments.

Performance
What is the code complexity?
The complexity is moderate, with multiple nested structures and loops that handle file operations and user interactions.

How does the complexity change with various inputs?
Complexity could increase with larger datasets due to the linear searches and file handling which may not scale well.

7. Why we submitted late.

We were on track to complete the project on time, but Saturday evening one of our members notified us that she was in a car accident. Due to this unexpected crisis, the other members had to divide her remaining work between the two of us. This additional workload delayed submission.