Serialization for the Parking System Application: Client/Server for

Master of Science

Information Technology

Bria Wright

University of Denver University College

May 25, 2025

Faculty: Nathan Braun MBA, BBA

Dean: Michael J. McGuire, MLS

Table of Contents

Introduction

In creating a parking system server application, I aimed to build a strong and flexible backend that could handle client requests, manage customer registrations, and handle car parking while providing clear responses. The server needed to understand incoming commands like "CUSTOMER" and "PARK," check the request data for accuracy, call the necessary business logic and send back standardized JSON responses to clients. I will discuss the design choices made while implementing the Server class. This covers how I handled requests, used JSON to format data, and developed our testing strategy. It also discusses the challenges I faced during development, such as managing JSON data with Java's Properties, directing commands correctly, and fixing compilation errors, as well as the techniques I used to solve these problems.

Design Overview

Core Server Responsibilities

The Server class is the central coordinator that processes client requests and generates responses. It is designed with the following responsibilities:

- Request Parsing: The server accepts ParkingRequest objects, which encapsulate a command string and a set of key-value properties.
- Command Routing: Based on the command, the server directs the request to the appropriate handler logic.
- Validation: Each command type requires validation of required fields to ensure data integrity.

- Response Generation: The server generates a ParkingResponse with a status code and message reflecting the outcome.
- JSON Serialization: Requests and responses are serialized and deserialized using Gson to facilitate network communication (Google 2024).

Command Processing Model

The server supports multiple commands, with "CUSTOMER" and "PARK" as primary examples:

- CUSTOMER Command: This command requires properties such as first name, last name, and email. The server validates these fields and simulates customer registration. On success, it returns a response with status code 0 and a success message. It returns an error status 1 and an informative error message on failure.
- PARK Command: This command requires properties like license and lot. It validates the data and simulates car parking, returning appropriate responses (Bloch 2017).

For unrecognized commands, the server returns an error response indicating the command is unknown (Bloch 2017).

Handling Properties with Java Properties

Since the client requests that key-value pairs be used for parameters, the design uses Java's Properties class to store these parameters. This allows easy property lookup by key. However, Properties is an older class that can be cumbersome compared to modern Map<String, String> types, especially with JSON serialization. To bridge this, the ParkingRequest class leverages Gson to serialize and deserialize JSON, carefully managing the Properties object (GeeksforGeeks 2017).

Testing Approach

Unit tests were written using JUnit 5 to verify the server's request handling correctness. Tests cover (Farcic and Garcia 2015):

- Successful customer registration.
- Customer registration failures due to missing fields.
- Successful car parking.
- Handling unknown commands gracefully.

This test-driven approach ensures the server behaves predictably under various inputs and helps catch regressions early.

Challenges and Solutions

Challenge 1: JSON Serialization of Properties

One of the initial hurdles encountered was managing the JSON serialization of the Properties object. This class extends `Hashtable<Object, Object>,` which presents a unique challenge for direct JSON serialization due to its use of generic objects as keys and values rather than the more typical string types. Unfortunately, Gson's default serialization behavior does not always navigate this complexity smoothly (Google 2024). To address this issue, the design retained the Properties object but ensured that only string keys and values were utilized, thus streamlining the serialization process. Furthermore, the ParkingRequest class was equipped with no-argument constructors, enhancing Gson's reflection-based serialization capabilities. Rigorous testing confirmed that the JSON serialization round-trip accurately preserved all fields, maintaining data integrity.

Challenge 2: Command Dispatch Without a Dedicated Handler Method

Then I anticipated implementing distinct methods, such as `handleRequest` or `handleCustomer,` for each command to manage command dispatching. However, due to the assignment and design requirements constraints, I needed to adopt a more consolidated approach using a single `handleRequest` method. I employed a streamlined command dispatching mechanism using a basic if-else structure within the `handleRequest` method to direct logic execution based on the command string (Gamma et al. 1994). While this method may lack the modular elegance of separate handler methods, it proved to be sufficiently effective for the limited command set available. Additionally, the system is designed to allow future refactoring into a more robust command pattern should the necessity arise (Gamma et al. 1994).

Challenge 3: Testing Without an Actual Network Server

Since the server was architected as a backend component without an operational socket server within the current scope, testing relied on simulating client-server interactions instead of network communication. I crafted unit tests that directly invoked the `handleRequest` method with meticulously constructed ParkingRequest objects, thus bypassing the need for networking altogether (Farcic and Garcia 2015). This approach simplified the testing process and allowed for focused validation of the core business logic. Furthermore, JSON serialization tests were implemented to ensure that the conversion processes between network messages functioned seamlessly and correctly.

Challenge 4: Balancing Simplicity and Extensibility

As this server project is fundamentally a learning exercise, I faced the challenge of balancing maintaining simplicity for readability and ensuring extensibility for future enhancements such as including additional commands, real database integration, or implementing a comprehensive networking layer. The Server class was thoughtfully structured to maintain a clear separation of concerns, and I made a point to document the code extensively to enhance maintainability. The ParkingRequest and ParkingResponse classes were designed to utilize JSON serialization, allowing for straightforward integration with genuine networking code down the line. Additionally, I produced comprehensive tests to guarantee robustness and reliability as the system evolves.

Conclusion

Implementing the Server class for the parking system project helped me improve my backend design, JSON serialization, command processing, and unit testing skills. I used a simple command routing model, Java's Properties for request parameters, and Gson for handling JSON. These choices effectively met the project requirements. I faced challenges with serialization quirks and design decisions but overcame them through research, testing, and refactoring. This taught me the importance of organizing classes and methods well, following Java conventions, and maintaining strong testing practices. Future enhancements include adding a real database, implementing asynchronous request handling, and refactoring command processing into a strategy or command pattern for improved modularity. Overall, this project strengthened my understanding of backend Java development and practical problem-solving in software design. I plan to implement asynchronous request handling and refactor command processing into a strategy or command pattern for future improvements. This will enhance the system’s modularity. Overall, this project has significantly improved my understanding of backend Java development and my problem-solving skills in software design.

References

Bloch, Joshua. 2017. *Effective Java, Third Edition*. Addison-Wesley Professional.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1994. “Design Patterns: Elements of Reusable Object Oriented Software.” https://www.grch.com.ar/docs/unlu.poo/Gamma-DesignPatternsIntro.pdf.

GeeksforGeeks. 2017. “Properties Class in Java.” GeeksforGeeks. July 11, 2017. https://www.geeksforgeeks.org/java-util-properties-class-java/#.

Google. 2024. "Gson User Guide." GitHub. https://github.com/google/gson/blob/main/UserGuide.md.

Farcic , Viktor and Garcia, Alex. 2015. *Test-Driven Java Development : Invoke TDD Principles for End-To-End Application Development with Java*. Birmingham: Packt Publishing, August.